METHOD AND SYSTEM FOR PRESENTATION AND SPECIFICATION OF DISTRIBUTED MULTI-CUSTOMER CONFIGURATION MANAGEMENT WITHIN A NETWORK MANAGEMENT FRAMEWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to an improved data processing system and, in particular, to a method and system for multiple computer or process coordinating. Still more particularly, the present invention provides a method and system for network management.

2. Description of Related Art

Technology expenditures have become a significant portion of operating costs for most enterprises, and businesses are constantly seeking ways to reduce information technology (IT) costs. This has given rise to an increasing number of outsourcing service providers, each promising, often contractually, to deliver reliable service while offloading the costly burdens of staffing, procuring, and maintaining an IT organization. While most service providers started as network pipe providers, they are moving into server outsourcing, application hosting, and desktop management. For those enterprises that do not outsource, they are demanding more accountability from their IT organizations as well as demanding that IT is integrated into their business goals. In both cases, "service level agreements" have been employed to contractually guarantee service delivery between an IT organization and its customers. As a

result, IT teams now require management solutions that focus on and support "business processes" and "service delivery" rather than just disk space monitoring and network pings.

IT solutions now require end-to-end management that includes network connectivity, server maintenance, and application management in order to succeed. The focus of IT organizations has turned to ensuring overall service delivery and not just the "towers" of network, server, desktop, and application. Management systems must fulfill two broad goals: a flexible approach that allows rapid deployment and configuration of new services for the customer; and an ability to support rapid delivery of the management tools themselves. A successful management solution fits into a heterogeneous environment, provides openness with which it can knit together management tools and other types of applications, and a consistent approach to managing all of the IT assets.

With all of these requirements, a successful management approach will also require attention to the needs of the staff within the IT organization to accomplish these goals: the ability of an IT team to deploy an appropriate set of management tasks to match the delegated responsibilities of the IT staff; the ability of an IT team to navigate the relationships and effects of all of their technology assets, including networks, middleware, and applications; the ability of an IT team to define their roles and responsibilities consistently and securely across the various management tasks; the ability of an IT team to define groups of customers and their services consistently across the

various management tasks; and the ability of an IT team to address, partition, and reach consistently the managed devices.

Many service providers have stated the need to be able to scale their capabilities to manage millions of devices. When one considers the number of customers in a home consumer network as well as pervasive devices, such as smart mobile phones, these numbers are quickly realized. Significant bottlenecks appear when typical IT solutions attempt to support more than several thousand devices.

Given such network spaces, a management system must be very resistant to failure so that service attributes, such as response time, uptime, and throughput, are delivered in accordance with guarantees in a service level agreement. In addition, a service provider may attempt to support many customers within a single network management system. The service provider's profit margins may materialize from the ability to bill usage of a common management system to multiple customers.

On the other hand, the service provider must be able to support contractual agreements on an individual basis. Service attributes, such as response time, uptime, and throughput, must be determinable for each customer. In order to do so, a network management system must provide a suite of network management tools that is able to perform device monitoring and discovery for each customer's network while integrating these abilities across a shared network backbone to gather the network management information into the service provider's distributed data processing system. There is a direct

relationship between the ability of a management system to provide network monitoring and discovery functionality and the ability of a service provider using the management system to serve multiple customers using a single management system. Preferably, the management system can replicate services, detect faults within a service, restart services, and reassign work to a replicated service. By implementing a common set of interfaces across all of their services, each service developer gains the benefits of system robustness. A well-designed, component-oriented, highly distributed system can easily accept a variety of services on a common infrastructure with built-in fault-tolerance and levels of service.

Given a scenario in which a service provider is using an integrated network management system for multiple customers, it is most likely that many different individuals will be assigned to manage different customers, different regions, and different groups of devices. In addition, separate individuals may have different duties within similar portions of the network, such as deploying new devices versus monitoring the uptime of those devices. In a highly distributed system comprising on the order of a million devices, the task of authenticating and authorizing the administrative actions of many individuals per customer, per region, per device, etc., becomes quite complex.

One significant issue with a large, multi-customer, management system is the manner in which the management system is to be configured and controlled. While a service provider may desire to perform most management

operations for all of its customers, some customers may desire to control certain aspects of the on-going management process because they prefer to have more control over physical assets and enterprise data.

Therefore, it would be particularly advantageous to provide a method and system within a network management framework for flexibly managing multiple customers within a highly distributed system. It would be particularly advantageous for the network management system to provide distributed configuration and control within a multi-customer environment.

SUMMARY OF THE INVENTION

A method, system, apparatus, and computer program product is presented for management of a distributed data processing system on behalf of a plurality of management customers. The distributed data processing system is logically represented as a set of scopes, wherein a scope is a logical organization of network-related objects. The network-related objects may be generated by dynamically discovering endpoints, systems, and networks within the distributed data processing system, which are then correspondingly represented as a set of endpoint objects, system objects, and network objects, any of which may be labeled as a network-related object. The endpoint objects, system objects, and network objects are logically organized into a set of scopes, and each endpoint object, each system object, and each network object is uniquely assigned to a scope such that scopes do not logically overlap. Each scope is uniquely assigned to a management customer. The distributed data processing system is managed as a set of logical networks in which a logical network contains a set of scopes and in which each logical network is uniquely assigned to a management customer. An administrative user may dynamically reconfigure the logical networks within the distributed data processing system while managing the logical networks for a set of customers.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, further objectives, and advantages thereof, will be best understood by reference to the following detailed description when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a diagram depicting a known logical configuration of software and hardware resources;

Figure 2A is simplified diagram illustrating a large distributed computing enterprise environment in which the present invention is implemented;

Figure 2B is a block diagram of a preferred system management framework illustrating how the framework functionality is distributed across the gateway and its endpoints within a managed region;

Figure 2C is a block diagram of the elements that comprise the low cost framework (LCF) client component of the system management framework;

Figure 2D is a diagram depicting a logical configuration of software objects residing within a hardware network similar to that shown in **Figure 2A**;

Figure 2E is a diagram depicting the logical relationships between components within a system management framework that includes two endpoints and a gateway;

Figure 2F is a diagram depicting the logical relationships between components within a system management framework that includes a gateway supporting two DKS-enabled applications;

Figure 2G is a diagram depicting the logical relationships between components within a system management framework that includes two gateways supporting two endpoints;

Figure 3 is a block diagram depicting components within the system management framework that provide resource leasing management functionality within a distributed computing environment such as that shown in **Figures 2D-2E**;

Figure 4 is a block diagram showing data stored by a the IPOP (IP Object Persistence) service;

Figure 5A is a block diagram showing the IPOP service in more detail;

Figure 5B is a network diagram depicting a set of routers that undergo a scoping process;

Figure 5C depicts the IP Object Security Hierarchy;

Figure 6 is a block diagram depicting a set of components that may be used to implement scope-based security access;

Figure 7A is a flowchart depicting a portion of an initialization process in which a network management system prepares a security subsystem for user access to network-related objects;

Figure 7B is a flowchart depicting further detail of the initialization process in which the DSC objects are initially created and stored;

Figure 7C is a flowchart depicting further detail of the initial DSC object creation process in which DSC objects are created and stored for an endpoint/user combination;

Figure 7D is a flowchart depicting further detail of the initial DSC object creation process in which DSC objects are created and stored for an endpoint/endpoint combination;

Figure 8A is a flowchart depicting a process for creating topology data;

Figure 8B is a flowchart depicting a process for listening for physical network changes that affect topology objects;

Figure 9A is a figure depicting a graphical user interface window that may be used by a network or system administrator to view the topology of a network that is being monitored;

Figure 9B is a graphical user interface window that shows the topology of a network that has changed;

Figure 10A is a block diagram depicting a known configuration of software and/or hardware network components linking multiple networks;

Figure 10B is a block diagram depicting a service provider connected to two customers that each have subnets that may contain duplicate network addresses;

Figure 11A is a block diagram showing a set of components that may be used to implement multi-customer management across multiple networks in which duplicate address may be present;

Figures 11B-11D are some simplified pseudo-code examples that depict an object-oriented manner in which action objects and endpoint objects can be implemented;

Figure 12A is a flowchart depicting a portion of an initialization process in which a network management system prepares for managing a set of networks with multiple NATs;

Figure 12B is a flowchart depicting further detail of the initialization process in which the administrator resolves addressability problems;

Figure 12C is a flowchart depicting further detail of the process in which the administrator assigns VPN IDs;

Figure 13 is a figure that depicts a graphical user interface (GUI) that may be used by a network or system administrator to set monitoring parameters for resolving address collisions;

Figure 14A is a flowchart showing the overall process for performing an IP "Ping" with in a multi-customer, distributed data processing system containing multiple private networks; and

Figure 14B is a flowchart that depicts a process by which an administrator chooses the source endpoint and target endpoint for the IP "Ping" action described in an overall manner in **Figure 14A;**

Figure 15 is a block diagram depicting a logical organization of network-related objects that may be managed within different scopes on behalf of multiple customers whose physical networks are being administered using the network management framework of the present invention;

Figure 16 is a flowchart depicting a process for handling certain types of scope-related events;

Figures 17A-17B are flowcharts depicting a process for relocating network-related objects based on an administrator-initiated change of scope or for configuring multiple customers within a distributed data processing system;

Figure 18A is a block diagram showing possible administrative applications to be used by administrative users for controlling the network management framework by configuring components and reviewing status via the administrative applications;

Figure 18B is a figure depicting a graphical user interface window that may be used by a network or system administrator to set the installation locations of various services;

Figure 19 is a flowchart depicting a process by which a multi-customer administrator may complete the first stage of a multi-customer installation;

Figure 20 is a figure depicting a graphical user interface window that may be used by a network or system administrator to configure some of the configuration parameters of the IPOP service;

Figures 21A-21B are a pair of flowcharts depicting a process by which a customer administrator within a multi-customer environment managed by the network management framework can perform certain management actions; and

Figures 22-33 depict graphical user interface windows that may be used to configure various aspects of the network management framework.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a methodology for managing a distributed data processing system. The manner in which the system management is performed is described further below in more detail after the description of the preferred embodiment of the distributed computing environment in which the present invention operates.

With reference now to **Figure 1**, a diagram depicts a known logical configuration of software and hardware resources. In this example, the software is organized in an object-oriented system. Application object **102**, device driver object **104**, and operating system object **106** communicate across network **108** with other objects and with hardware resources **110-114**.

In general, the objects require some type of processing, input/output, or storage capability from the hardware resources. The objects may execute on the same device to which the hardware resource is connected, or the objects may be physically dispersed throughout a distributed computing environment. The objects request access to the hardware resource in a variety of manners, e.g. operating system calls to device drivers. Hardware resources are generally available on a first-come, first-serve basis in conjunction with some type of arbitration scheme to ensure that the requests for resources are fairly handled. In some cases, priority may be given to certain requesters, but in most

implementations, all requests are eventually processed.

With reference now to **Figure 2A**, the present invention is preferably implemented in a large distributed computer environment **210** comprising up to thousands of "nodes". The nodes will typically be geographically dispersed and the overall environment is "managed" in a distributed manner. Preferably, the managed environment is logically broken down into a series of loosely connected managed regions (MRs) **212**, each with its own management server **214** for managing local resources with the managed region. The network typically will include other servers (not shown) for carrying out other distributed network functions. These include name servers, security servers, file servers, thread servers, time servers and the like. Multiple servers **214** coordinate activities across the enterprise and permit remote management and operation. Each server **214** serves a number of gateway machines **216**, each of which in turn support a plurality of endpoints/terminal nodes **218**. The server **214** coordinates all activity within the managed region using a terminal node manager at server **214**.

With reference now to **Figure 2B**, each gateway machine **216** runs a server component **222** of a system management framework. The server component **222** is a multi-threaded runtime process that comprises several components: an object request broker (ORB) **221**, an authorization service **223**, object location service **225** and basic object adapter (BOA) **227**. Server component **222** also includes an object library **229**. Preferably, ORB **221**

runs continuously, separate from the operating system, and it communicates with both server and client processes through separate stubs and skeletons via an interprocess communication (IPC) facility **219**. In particular, a secure remote procedure call (RPC) is used to invoke operations on remote objects. Gateway machine **216** also includes operating system **215** and thread mechanism **217**.

The system management framework, also termed distributed kernel services (DKS), includes a client component **224** supported on each of the endpoint machines **218**. The client component **224** is a low cost, low maintenance application suite that is preferably "dataless" in the sense that system management data is not cached or stored there in a persistent manner. Implementation of the management framework in this "client-server" manner has significant advantages over the prior art, and it facilitates the connectivity of personal computers into the managed environment. It should be noted, however, that an endpoint may also have an ORB for remote object-oriented operations within the distributed environment, as explained in more detail further below.

Using an object-oriented approach, the system management framework facilitates execution of system management tasks required to manage the resources in the managed region. Such tasks are quite varied and include, without limitation, file and data distribution, network usage monitoring, user management, printer or other resource configuration management, and the like. In a preferred implementation, the object-oriented framework includes a Java runtime environment for well-known

advantages, such as platform independence and standardized interfaces. Both gateways and endpoints operate portions of the system management tasks through cooperation between the client and server portions of the distributed kernel services.

In a large enterprise, such as the system that is illustrated in **Figure 2A**, there is preferably one server per managed region with some number of gateways. For a workgroup-size installation, e.g., a local area network, a single server-class machine may be used as both a server and a gateway. References herein to a distinct server and one or more gateway(s) should thus not be taken by way of limitation as these elements may be combined into a single platform. For intermediate size installations, the managed region grows breadth-wise, with additional gateways then being used to balance the load of the endpoints.

The server is the top-level authority over all gateways and endpoints. The server maintains an endpoint list, which keeps track of every endpoint in a managed region. This list preferably contains all information necessary to uniquely identify and manage endpoints including, without limitation, such information as name, location, and machine type. The server also maintains the mapping between endpoints and gateways, and this mapping is preferably dynamic.

As noted above, there are one or more gateways per managed region. Preferably, a gateway is a fully managed node that has been configured to operate as a gateway. In certain circumstances, though, a gateway may be regarded as an endpoint. A gateway always has a network

interface card (NIC), so a gateway is also always an endpoint. A gateway usually uses itself as the first seed during a discovery process. Initially, a gateway does not have any information about endpoints. As endpoints login, the gateway builds an endpoint list for its endpoints. The gateway's duties preferably include: listening for endpoint login requests, listening for endpoint update requests, and (its main task) acting as a gateway for method invocations on endpoints.

As also discussed above, the endpoint is a machine running the system management framework client component, which is referred to herein as a management agent. The management agent has two main parts as illustrated in **Figure 2C**: daemon **226** and application runtime library **228**. Daemon **226** is responsible for endpoint login and for spawning application endpoint executables. Once an executable is spawned, daemon **226** has no further interaction with it. Each executable is linked with application runtime library **228**, which handles all further communication with the gateway.

Each endpoint is also a computing device. In one preferred embodiment of the invention, most of the endpoints are personal computers, e.g., desktop machines or laptops. In this architecture, the endpoints need not be high powered or complex machines or workstations. An endpoint computer preferably includes a Web browser such as Netscape Navigator or Microsoft Internet Explorer. An endpoint computer thus may be connected to a gateway via the Internet, an intranet, or some other computer network.

Preferably, the client-class framework running on

each endpoint is a low-maintenance, low-cost framework that is ready to do management tasks but consumes few machine resources because it is normally in an idle state. Each endpoint may be "dataless" in the sense that system management data is not stored therein before or after a particular system management task is implemented or carried out.

With reference now to **Figure 2D**, a diagram depicts a logical configuration of software objects residing within a hardware network similar to that shown in **Figure 2A**. The endpoints in **Figure 2D** are similar to the endpoints shown in **Figure 2B**. Object-oriented software, similar to the collection of objects shown in **Figure 1**, executes on the endpoints. Endpoints **230** and **231** support application action object **232** and application object **233**, device driver objects **234-235**, and operating system objects **236-237** that communicate across a network with other objects and hardware resources.

Resources can be grouped together by an enterprise into managed regions representing meaningful groups. Overlaid on these regions are domains that divide resources into groups of resources that are managed by gateways. The gateway machines provide access to the resources and also perform routine operations on the resources, such as polling. **Figure 2D** shows that endpoints and objects can be grouped into managed regions that represent branch offices **238** and **239** of an enterprise, and certain resources are controlled by central office **240**. Neither a branch office nor a central office is necessarily restricted to a single

physical location, but each represents some of the hardware resources of the distributed application framework, such as routers, system management servers, endpoints, gateways, and critical applications, such as corporate management Web servers. Different types of gateways can allow access to different types of resources, although a single gateway can serve as a portal to resources of different types.

With reference now to **Figure 2E**, a diagram depicts the logical relationships between components within a system management framework that includes two endpoints and a gateway. **Figure 2E** shows more detail of the relationship between components at an endpoint. Network **250** includes gateway **251** and endpoints **252** and **253**, which contain similar components, as indicated by the similar reference numerals used in the figure. An endpoint may support a set of applications **254** that use services provided by the distributed kernel services **255**, which may rely upon a set of platform-specific operating system resources **256**. Operating system resources may include TCP/IP-type resources, SNMP-type resources, and other types of resources. For example, a subset of TCP/IP-type resources may be a line printer (LPR) resource that allows an endpoint to receive print jobs from other endpoints. Applications **254** may also provide self-defined sets of resources that are accessible to other endpoints. Network device drivers **257** send and receive data through NIC hardware **258** to support communication at the endpoint.

With reference now to **Figure 2F**, a diagram depicts

the logical relationships between components within a system management framework that includes a gateway supporting two DKS-enabled applications. Gateway **260** communicates with network **262** through NIC **264**. Gateway **260** contains ORB **266** that supports DKS-enabled applications **268** and **269**. **Figure 2F** shows that a gateway can also support applications. In other words, a gateway should not be viewed as merely being a management platform but may also execute other types of applications.

With reference now to **Figure 2G**, a diagram depicts the logical relationships between components within a system management framework that includes two gateways supporting two endpoints. Gateway **270** communicates with network **272** through NIC **274**. Gateway **270** contains ORB **276** that may provide a variety of services, as is explained in more detail further below. In this particular example, **Figure 2G** shows that a gateway does not necessarily connect with individual endpoints.

Gateway **270** communicates through NIC **278** and network **279** with gateway **280** and its NIC **282**. Gateway **280** contains ORB **284** for supporting a set of services. Gateway **280** communicates through NIC **286** and network **287** to endpoint **290** through its NIC **292** and to endpoint **294** through its NIC **296**. Endpoint **290** contains ORB **298** while endpoint **294** does not contain an ORB. In this particular example, **Figure 2G** also shows that an endpoint does not necessarily contain an ORB. Hence, any use of endpoint **294** as a resource is performed solely through management processes at gateway **280**.

Figures 2F and **2G** also depict the importance of gateways in determining routes/data paths within a highly distributed system for addressing resources within the system and for performing the actual routing of requests for resources. The importance of representing NICs as objects for an object-oriented routing system is described in more detail further below.

As noted previously, the present invention is directed to a methodology for managing a distributed computing environment. A resource is a portion of a computer system's physical units, a portion of a computer system's logical units, or a portion of the computer system's functionality that is identifiable or addressable in some manner to other physical or logical units within the system.

With reference now to **Figure 3**, a block diagram depicts components within the system management framework within a distributed computing environment such as that shown in **Figures 2D-2E**. A network contains gateway **300** and endpoints **301** and **302**. Gateway **302** runs ORB **304**. In general, an ORB can support different services that are configured and run in conjunction with an ORB. In this case, distributed kernel services (DKS) include Network Endpoint Location Service (NELS) **306**, IP Object Persistence (IPOP) service **308**, and gateway service **310**.

The gateway service processes action objects, which are explained in more detail below, and directly communicates with endpoints or agents to perform management operations. The gateway receives events from resources and passes the events to interested parties within the distributed system. The NELS works in

combination with action objects and determines which gateway to use to reach a particular resource. A gateway is determined by using the discovery service of the appropriate topology driver, and the gateway location may change due to load balancing or failure of primary gateways.

Other resource level services may include an SNMP (Simple Network Management Protocol) service that provides protocol stacks, polling service, and trap receiver and filtering functions. The SNMP service can be used directly by certain components and applications when higher performance is required or the location independence provided by the gateways and action objects is not desired. A metadata service can also be provided to distribute information concerning the structure of SNMP agents.

The representation of resources within DKS allows for the dynamic management and use of those resources by applications. DKS does not impose any particular representation, but it does provide an object-oriented structure for applications to model resources. The use of object technology allows models to present a unified appearance to management applications and hide the differences among the underlying physical or logical resources. Logical and physical resources can be modeled as separate objects and related to each other using relationship attributes.

By using objects, for example, a system may implement an abstract concept of a router and then use this abstraction within a range of different router hardware. The common portions can be placed into an

abstract router class while modeling the important differences in subclasses, including representing a complex system with multiple objects. With an abstracted and encapsulated function, the management applications do not have to handle many details for each managed resource. A router usually has many critical parts, including a routing subsystem, memory buffers, control components, interfaces, and multiple layers of communication protocols. Using multiple objects has the burden of creating multiple object identifiers (OIDs) because each object instance has its own OID. However, a first order object can represent the entire resource and contain references to all of the constituent parts.

Each endpoint may support an object request broker, such as ORBs **320** and **322**, for assisting in remote object-oriented operations within the DKS environment. Endpoint **301** contains DKS-enabled application **324** that utilizes object-oriented resources found within the distributed computing environment. Endpoint **302** contains target resource provider object or application **326** that services the requests from DKS-enabled application **324**. A set of DKS services **330** and **334** support each particular endpoint.

Applications require some type of insulation from the specifics of the operations of gateways. In the DKS environment, applications create action objects that encapsulate commands which are sent to gateways, and the applications wait for the return of the action object. Action objects contain all of the information necessary to run a command on a resource. The application does not need to know the specific protocol that is used to

communicate with the resource. The application is unaware of the location of the gateway because it issues an action object into the system, and the action object itself locates and moves to the correct gateway. The location independence allows the NELS to balance the load between gateways independently of the applications and also allows the gateways to handle resources or endpoints that move or need to be serviced by another gateway.

The communication between a gateway and an action object is asynchronous, and the action objects provide error handling and recovery. If one gateway goes down or becomes overloaded, another gateway is located for executing the action object, and communication is established again with the application from the new gateway. Once the controlling gateway of the selected endpoint has been identified, the action object will transport itself there for further processing of the command or data contained in the action object. If it is within the same ORB, it is a direct transport. If it is within another ORB, then the transport can be accomplished with a "Moveto" command or as a parameter on a method call.

Queuing the action object on the gateway results in a controlled process for the sending and receiving of data from the IP devices. As a general rule, the queued action objects are executed in the order that they arrive at the gateway. The action object may create child action objects if the collection of endpoints contains more than a single ORB ID or gateway ID. The parent action object is responsible for coordinating the completion status of any of its children. The creation

of child action objects is transparent to the calling application. A gateway processes incoming action objects, assigns a priority, and performs additional security challenges to prevent rogue action object attacks. The action object is delivered to the gateway that must convert the information in the action object to a form suitable for the agent. The gateway manages multiple concurrent action objects targeted at one or more agents, returning the results of the operation to the calling application as appropriate.

In the preferred embodiment, potentially leasable target resources are Internet protocol (IP) commands, e.g. pings, and Simple Network Management Protocol (SNMP) commands that can be executed against endpoints in a managed region. Referring again to **Figures 2F** and **2G**, each NIC at a gateway or an endpoint may be used to address an action object. Each NIC is represented as an object within the IPOP database, which is described in more detail further below.

The Action Object IP (AOIP) Class is a subclass of the Action Object Class. An AOIP object is the primary vehicle that establishes a connection between an application and a designated IP endpoint using a gateway or stand-alone service. In addition, the Action Object SNMP (AOSnmp) Class is also a subclass of the Action Object Class. An AOSnmp object is the primary vehicle that establishes a connection between an application and a designated SNMP endpoint via a gateway or the gateway service. However, the present invention is primarily concerned with IP endpoints.

The AOIP class should include the following: a

constructor to initialize itself; an interface to the NELS; a mechanism by which the action object can use the ORB to transport itself to the selected gateway; a security check verification of access rights to endpoints; a container for either data or commands to be executed at the gateway; a mechanism by which to pass commands or classes to the appropriate gateway or endpoint for completion; and public methods to facilitate the communication between objects.

The instantiation of an AOIP object creates a logical circuit between an application and the targeted gateway or endpoint. This circuit is persistent until command completion through normal operation or until an exception is thrown. When created, the AOIP object instantiates itself as an object and initializes any internal variables required. An AOIP may be capable of running a command from inception or waiting for a future command. A program that creates an AOIP object must supply the following elements: address of endpoints; function to be performed on the endpoint; and data arguments specific to the command to be run. A small part of the action object must contain the return end path for the object. This may identify how to communicate with the action object in case of a breakdown in normal network communications. An action object can contain either a class or object containing program information or data to be delivered eventually to an endpoint or a set of commands to be performed at the appropriate gateway. Action objects IP return back a result for each address endpoint targeted.

Using commands such as "Ping", "Trace Route",

"Wake-On LAN", and "Discovery", the AOIP object performs the following services: facilitates the accumulation of metrics for the user connections; assists in the description of the topology of a connection; performs Wake-On LAN tasks using helper functions; and discovers active agents in the network environment.

The NELS service finds a route to communicate between the application and the appropriate endpoint. The NELS service converts input to protocol, network address, and gateway location for use by action objects. The NELS service is a thin service that supplies information discovered by the IPOP service. The primary roles of the NELS service are as follows: support the requests of applications for routes; maintain the gateway and endpoint caches that keep the route information; ensure the security of the requests; and perform the requests as efficiently as possible to enhance performance.

For example, an application requires a target endpoint (target resource) to be located. The target is ultimately known within the DKS space using traditional network values, i.e. a specific network address and a specific protocol identifier. An action object is generated on behalf of an application to resolve the network location of an endpoint. The action object asks the NELS service to resolve the network address and define the route to the endpoint in that network.

One of the following is passed to the action object to specify a destination endpoint: an EndpointAddress object; a fully decoded NetworkAddress object; or a string representing the IP address of the IP endpoint.

In combination with the action objects, the NELS service determines which gateway to use to reach a particular resource. The appropriate gateway is determined using the discovery service of the appropriate topology driver and may change due to load balancing or failure of primary gateways. An "EndpointAddress" object must consist of a collection of at least one or more unique managed resource IDs. A managed resource ID decouples the protocol selection process from the application and allows the NELS service to have the flexibility to decide the best protocol to reach an endpoint. On return from the NELS service, an "AddressEndpoint" object is returned, which contains enough information to target the best place to communicate with the selected IP endpoints. It should be noted that the address may include protocol-dependent addresses as well as protocol-independent addresses, such as the virtual private network id and the IPOP Object ID. These additional addresses handle the case where duplicate addresses exist in the managed region.

When an action needs to be taken on a set of endpoints, the NELS service determines which endpoints are managed by which gateways. When the appropriate gateways are identified, a single copy of the action object is distributed to each identified gateway. The results from the endpoints are asynchronously merged back to the caller application through the appropriate gateways. Performing the actions asynchronously allows for tracking all results whether the endpoints are connected or disconnected. If the action object IP fails to execute on its target gateway, NELS is consulted to

identify an alternative path for the command. If an alternate path is found, the action object IP is transported to that gateway and executed. It may be assumed that the entire set of commands within one action object IP must fail before this recovery procedure is invoked.

With reference now to **Figure 4**, a block diagram shows the manner in which data is stored by the IPOP (IP Object Persistence) service. IPOP service database **402** contains endpoint database table **404**, system database table **406**, and network database table **408**. Each table contains a set of topological objects (TopoObjects) for facilitating the leasing of resources at IP endpoints and the execution of action objects. Information within IPOP service database **402** allows applications to generate action objects for resources previously identified as IP objects through a discovery process across the distributed computing environment. **Figure 4** merely shows that the TopoObjects may be separated into a variety of categories that facilitate processing on the various objects. The separation of physical network categories facilitates the efficient querying and storage of these objects while maintaining the physical network relationships in order to produce a graphical user interface of the network topology.

Ever since computers have been connected in networks, there has been a need for mechanisms to discover and monitor the network devices. Today, with computer power being pushed down to the smallest devices, networks have grown larger. It is not unreasonable to assume that soon networks will frequently comprise

millions or even tens of millions of systems.

Network use has also changed in recent years. Autonomous networks, i.e. those which are owned, managed and used by a single enterprise, are being replaced by "service provider" environments. In a service provider environment, a business's network resources (hardware and/or software) are contracted from a third party. The network is managed by that service provider for a number of clients who may, in fact, be transparently sharing resources.

Given the evolution of networked computing, it is obvious that the software used to discover and monitor devices must be scalable, reliable, flexible and secure. The most basic functionality of the DKS network management framework is to query the network to find which machines are present, how they are connected, and then store that data, as well as some status data, for later retrieval by higher-level applications, such as network topology graphers, administrative consoles, and the like. The DKS framework can be configured to provide a "snapshot" of the configuration or to poll the network and update data with changing configuration.

The DKS framework is a distributed system composed of one or more individual IP drivers which are independently configured, although they may share common configuration parameters. Implementation in Java or another platform-neutral language increases portability, although a small amount of native (platform-specific) code is required for the base communications stacks.

Configuration parameters include, but are not limited to, a unique numeric identifier used to couple

data stored in the IPOP centralized database with its owning IP driver, discovery mechanisms, discovery scope (subnets, subnet masks, private network identifiers), masks to limit the discovery within a defined scope, polling intervals, and limits on the number of data items stored in local in-memory cache. A set of default parameters, stored in a centralized location, is available for all IP drivers and each parameter may be overridden by local configuration.

Clients bind to the DKS framework either non-preferentially (in the case of non-specific service requests, such as listing or updating the default configuration parameters) or preferentially (in the case of specific service requests, such as listing the system information for a particular machine or updating the configuration parameters for a particular IP driver). This binding can be transparent if the DKS framework is part of a larger distributed computing environment. For example, if each IP driver in the DKS framework is launched within an ORB (Object Request Broker), the ORB functionality will provide location transparency. Similarly, if DKS framework is a service within the Distributed Computing Environment (DCE), each IP driver can register itself with the CDS (Cell Directory Service) which then provides clients with the proper IP driver binding based on location, data required, service name or other parameters.

In the absence of an encompassing distributed computing framework, clients could communicate to with the IP driver on a system via a well-known port or communication channel and the IP drivers could

communicate with one another via private communication channels. When an IP driver receives a request that should be handled by a different IP driver, it transfers the request to its peer.

As stated, the primary function of each IP driver is to discover and optionally monitor the subnets that are assigned to it at configuration time. Different network discovery mechanisms are supported and any or all may be used, depending on the environment and data needs. Supported discovery mechanisms include the following.

Route Table Discovery: This type of discovery defines a network in terms of connectivity between subnets, rather than an enumeration of all machines in the network. Route table discovery starts with discovery of known router (a multi-homed system that receives packets from one subnet and forwards them, as appropriate, to systems on another subnet). If that router falls within scope, it is "pinged" (Packet Internetwork Groper), a TCP/IP protocol commonly used to test connectivity between systems) to verify it is reachable and then an SNMP (Simple Network Management Protocol) query is issued to extract the route table. For each entry in the route table, if that entry falls within the IP driver's scope, it is then discovered and its route table is extracted, and discovery continues in that manner. This type of discovery is suitable when only a minimum of system information is needed to show basic connectivity in the network.

ARP (Address Resolution Protocol) Table Discovery:

This type of discovery starts with a "seed" system. If that seed falls into the scope of the IP driver, the

IP driver will ping the system to determine if it is up or down. If the system is up, the IP driver will issue SNMP requests to obtain system information. Included in this information is the contents of the seed system's ARP table, which contains the addresses of other systems which the seed has recently been in contact with. The IP driver then discovers any system from the ARP table that falls within scope, obtains those systems' ARP tables and continues discovery in that manner. This discovery mechanism finds more systems than route table discovery, but fewer than ping spread discovery (described below).

Ping Spread Discovery: This type of discovery attempts to find every system in a subnet by issuing a TCP/IP "ping" request to each system in the subnet. If a system responds, then the IP driver optionally may send additional SNMP requests to get system status information. This type of discovery could be used if a very accurate account of all systems is necessary, but should not be used in environments in which network traffic must be minimized for reasons of performance or cost (for example, leased line situations).

After discovery, the IP driver will store network, system and endpoint data in persistent data (which is typically a commercial database, but may be as simple as file-based data store). For performance reasons, a fraction of the data is stored in in-memory cache, the size of which can be specified at configuration. When an IP driver is stopped and restarted, it can load its cache from persistent data.

If network monitoring is required in addition to network discovery, a pool of threads (the number of which

is configurable) are used to poll individual systems and endpoints at configured polling intervals. The polled data is then committed to in-memory and persistent data store.

In today's service provider environment, "24/7" availability is often written into contract agreements; fault tolerance has become a critical feature. As previously noted, the DKS framework can be incorporated into a ORB or distributed computing environment. In those circumstances, the encompassing system can provide a heartbeat poll to each IP driver to ensure that it is up. If the IP driver fails to respond, the ORB's failover service can start a new IP driver up on a different system or ORB to automatically assume the responsibilities of the IP driver that failed.

In the absence of an encompassing distributed computing framework, one IP driver in the DKS framework could be the "master" IP driver and be responsible for assuring that all IP drivers are available and for the transfer of responsibilities in the event of failure. In the event of the master's failure any of a number of election algorithms could be used to determine a new master from the remaining IP drivers. Alternatively, each IP driver could have a "buddy" whose responsibilities are assumed in the event of failure. In either failover implementation, the "buddy" system must have the same security controls as the original to prevent unauthorized data access.

Security (access control) can be provided at multiple levels: at the DKS framework as a whole, at individual IP drivers, and at network object level

(networks, systems and endpoints). This way, multiple customers can use the same machine, the same DKS framework or even the same IP driver and still be restricted to accessing only the network data to which they are authorized.

Access control depends on two things: user authentication and user authorization. User authentication can be a function of the native operating system or can be an add-on authentication step at the application level (for example, DCE's dce_login). Regardless of the source of authentication, some kind of security context or token is created at login and must accompany each service request.

Authorization can be defined by attaching access control lists ("who can do what") or required capabilities to the object (or object category) in question. Capabilities are granted to users and are based on security configuration. When a user authenticates (logs in), his capabilities are stored in his security context. Authorization data can be implemented in different locations, for example, a database or the java.policy file.

When a client requests a DKS framework service, the security context is examined for the user identity and/or capabilities. If the user does not have required access to the DKS framework as a whole or the specific IP driver to which he must communicate, the service request is rejected. If the user does have access to the IP driver (or the entire DKS framework), the authorization is checked against the specific object that is being accessed, if there is one. If the user has access to that

object, the service request is honored, if not, the request is rejected.

With reference now to **Figure 5A**, a block diagram shows the IPOP service in more detail. In the preferred embodiment of the present invention, an IP driver subsystem is implemented as a collection of software components for discovering, i.e. detecting, IP "objects", i.e. IP networks, IP systems, and IP endpoints by using physical network connections. This discovered physical network is used to create topology data that is then provided through other services via topology maps accessible through a graphical user interface (GUI) or for the manipulation of other applications. The IP driver system can also monitor objects for changes in IP topology and update databases with the new topology information. The IPOP service provides services for other applications to access the IP object database.

IP driver subsystem **500** contains a conglomeration of components, including one or more IP drivers **502**. Every IP driver manages its own "scope", which is described in more detail further below, and every IP driver is assigned to a topology manager within topology service **504**, which can serve more than one IP driver. Topology service **504** stores topology information obtained from discovery controller **506**. The information stored within the topology service may include graphs, arcs, and the relationships between nodes determined by IP mapper **508**. Users can be provided with a GUI to navigate the topology, which can be stored within a database within the topology service.

IPOP service **510** provides a persistent repository

512 for discovered IP objects; persistent repository **512** contains attributes of IP objects without presentation information. Discovery controller **506** detects IP objects in Physical IP networks **514**, and monitor controller **516** monitors IP objects. A persistent repository, such as IPOP database **512**, is updated to contain information about the discovered and monitored IP objects. IP driver may use temporary IP data store component **518** and IP data cache component **520** as necessary for caching IP objects or storing IP objects in persistent repository **512**, respectively. As discovery controller **506** and monitor controller **516** perform detection and monitoring functions, events can be written to network event manager application **522** to alert network administrators of certain occurrences within the network, such as the discovery of duplicate IP addresses or invalid network masks.

External applications/users **524** can be other users, such as network administrators at management consoles, or applications that use IP driver GUI interface **526** to configure IP driver **502**, manage/unmanage IP objects, and manipulate objects in persistent repository **512**. Configuration service **528** provides configuration information to IP driver **502**. IP driver controller **530** serves as central control of all other IP driver components.

Referring back to **Figure 2G**, a network discovery engine is a distributed collection of IP drivers that are used to ensure that operations on IP objects by gateways **260**, **270**, and **280** can scale to a large installation and

provide fault-tolerant operation with dynamic start/stop or reconfiguration of each IP driver. The IPOP service stores and retrieves information about discovered IP objects; to do so, the IPOP service uses a distributed database in order to efficiently service query requests by a gateway to determine routing, identity, or a variety of details about an endpoint. The IPOP service also services queries by the topology service in order to display a physical network or map them to a logical network, which is a subset of a physical network that is defined programmatically or by an administrator. IPOP fault tolerance is also achieved by distribution of IPOP data and the IPOP service among many Endpoint ORBs.

One or more IP drivers can be deployed to provide distribution of IP discovery and promote scalability of IP driver subsystem services in large networks where a single IP driver is not sufficient to discover and monitor all IP objects. Each IP driver performs discovery and monitoring on a collection of IP resources within the driver's "scope". A driver's scope, which is explained in more detail below, is simply the set of IP subnets for which the driver is responsible for discovering and monitoring. Network administrators generally partition their networks into as many scopes as needed to provide distributed discovery and satisfactory performance.

A potential risk exists if the scope of one driver overlaps the scope of another, i.e. if two drivers attempt to discover/monitor the same device. Accurately defining unique and independent scopes may require the development of a scope configuration tool to verify the

uniqueness of scope definitions. Routers also pose a potential problem in that while the networks serviced by the routers will be in different scopes, a convention needs to be established to specify to which network the router "belongs", thereby limiting the router itself to the scope of a single driver.

Some ISPs may have to manage private networks whose addresses may not be unique across the installation, like 10.0.0.0 network. In order to manage private networks properly, first, the IP driver has to be installed inside the internal networks in order to be able to discover and manage the networks. Second, since the discovered IP addresses may not be unique across an entire installation that consists of multiple regions, multiple customers, etc., a private network ID has to be assigned to the private network addresses. In the preferred embodiment, the unique name of a subnet becomes "privateNetworkId\subnetAddress". Those customers that do not have duplicate networks address can just ignore the private network ID; the default private network ID is 0.

If Network Address Translator (NAT) is installed to translate the internal IP addresses to Internet IP addresses, users can install the IP drivers outside of NAT and manage the IP addresses inside the NAT. In this case, an IP driver will see only the translated IP addresses and discover only the IP addresses translated. If not all IP addresses inside the NAT are translated, an IP driver will not able to discover all of them. However, if IP drivers are installed this way, users do not have to configure the private network within the IP driver's scope.

Scope configuration is important to the proper operation of the IP drivers because IP drivers assume that there are no overlaps in the drivers' scopes. Since there should be no overlaps, every IP driver has complete control over the objects within its scope. A particular IP driver does not need to know anything about the other IP drivers because there is no synchronization of information between IP drivers. The configuration service provides the means to allow the DKS components to store and retrieve configuration information for a variety of other services from anywhere in the networks. In particular, the scope configuration will be stored in the configuration services so that IP drivers and other applications can access the information.

The ranges of addresses that a driver will discover and monitor are determined by associating a subnet address with a subnet mask and associating the resulting range of addresses with a subnet priority. An IP driver is a collection of such ranges of addresses, and the subnet priority is used to help decide the system address. A system can belong to two or more subnets, such as is commonly seen with a Gateway. The system address is the address of one of the NICs that is used to make SNMP queries. A user interface can be provided, such as an administrator console, to write scope information into the configuration service. System administrators do not need to provide this information at all, however, as the IP drivers can use default values.

An IP driver gets its scope configuration information from the configuration service, which may be stored using the following format:

scopeID=driverID,anchorname,subnetAddress:subnetMask[
:privateNetworkId:privateNetworkName:subnetPriority][,
subnetAddress:subnetMask:privateNetworkId:privateNetworkN
ame:subnetPriority]]

Typically, one IP driver manages only one scope. Hence, the "scopeID" and "driverID" would be the same. However, the configuration can provide for more than one scope managed by the same driver. "Anchorname" is the name in the name space in which the topology service will put the IP driver's network objects.

A scope does not have to include an actual subnet configured in the network. Instead, users/administrators can group subnets into a single, logical scope by applying a bigger subnet mask to the network address. For example, if a system has subnet "147.0.0.0" with mask of "255.255.0.0" and subnet "147.1.0.0" with a subnet mask of "255.255.0.0", the subnets can be grouped into a single scope by applying a mask of "255.254.0.0". Assume that the following table is the scope of IP Driver 2. The scope configuration for IP Driver 2 from the configuration service would be:
2=2,ip,147.0.0.0:255.254.0.0,146.100.0.0:255.255.0.0,
69.0.0.0:255.0.0.0.

Subnet address	Subnet mask
147.0.0.0	255.255.0.0
147.1.0.0	255.255.0.0
146.100.0.0	255.255.0.0
69.0.0.0	255.0.0.0

In general, an IP system is associated with a single IP address, and the "scoping" process is a straightforward association of a driver's ID with the system's IP address.

Routers and multi-homed systems, however, complicate the discovery and monitoring process because these devices may contain interfaces that are associated with different subnets. If all subnets of routers and multi-homed systems are in the scope of the same driver, the IP driver will manage the whole system. However, if the subnets of routers and multi-homed systems are across the scopes of different drivers, a convention is needed to determine a dominant interface: the IP driver that manages the dominant interface will manage the router object so that the router is not being detected and monitored by multiple drivers; each interface is still managed by the IP driver determined by its scope; the IP address of the dominant interface will be assigned as the system address of the router or multi-homed system; and the smallest (lowest) IP address of any interface on the router will determine which driver includes the router object within its scope.

Users can customize the configuration by using the subnet priority in the scope configuration. The subnet priority will be used to determinate the dominant interface before using the lowest IP address. If the subnet priorities are the same, the lowest IP address is then used. Since the default subnet priority would be "0", then the lowest IP address would be used by default.

With reference now to **Figure 5B**, a network diagram

depicts a network with a router that undergoes a scoping process. IP driver D1 will include the router in its scope because the subnet associated with that router interface is lower than the other three subnet addresses. However, each driver will still manage those interfaces inside the router in its scope. Drivers D2 and D3 will monitor the devices within their respective subnets, but only driver D1 will store information about the router itself in the IPOP database and the topology service database.

If driver D1's entire subnet is removed from the router, driver D2 will become the new "owner" of the router object because the subnet address associated with driver D2 is now the lowest address on the router. Because there is no synchronization of information between the drivers, the drivers will self-correct over time as they periodically rediscover their resources. When the old driver discovers that it no longer owns the router, it deletes the router's information from the databases. When the new driver discovers the router's lowest subnet address is now within its scope, the new driver takes ownership of the router and updates the various databases with the router's information. If the new driver discovers the change before the old driver has deleted the object, then the router object may be briefly represented twice until the old owner deletes the original representation.

There are two kinds of associations between IP objects. One is "IP endpoint in IP system" and the other is "IP endpoint in IP network". The implementation of associations relies on the fact that an IP endpoint has

the object IDs (OIDs) of the IP system and the IP network in which it is located. An IP driver can partition all IP networks, IP Systems, and IP endpoints into different scopes. A network and all its IP endpoints will always be assigned in the same scope. However, a router may be assigned to an IP driver, but some of its interfaces are assigned to different IP drivers. The IP drivers that do not manage the router but manage some of its interfaces will have to create interfaces but not the router object. Since those IP drivers do not have a router object ID to assign to its managed interfaces, they will assign a unique system name instead of object ID in the IP endpoint object to provide a link to the system object in a different driver.

Because of the inter-scope association, when the IP Object Persistence Service (IPOP) is queried to find all the IP endpoints in system, it will have to search not only IP endpoints with the system ID but also IP endpoints with its system name. If a distributed IP Object Persistence Service is implemented, the service has to provide extra information for searching among its distributed instances.

An IP driver may use a security service to check access to IP objects. In order to handle large number of objects, the security service requires the users to provide a naming hierarchy as the grouping mechanism. **Figure 5C**, described below, shows a security naming hierarchy of IP objects. An IP driver has to allow users to provide security down to the object level and to achieve high performance. In order to achieve this goal, the concepts of "anchor" and "unique object name" are

introduced. An anchor is a name in the naming space which can be used to plug in IP networks. Users can define, under the anchor, scopes that belong to the same customer or to a region. The anchor is then used by the security service to check if a user has access to the resource under the anchor. If users want a security group defined inside a network, the unique object name is used. A unique object name is in the format of:

IP network - privateNetworkID/binaryNetworkAddress

IP system - privateNetworkID/binaryIPAddress/system

IP endpoint-
privateNetworkID/binaryNetworkAddress/endpoint

For example:

A network "146.84.28.0:255.255.255.0" in privateNetworkID 12 has unique name:
12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0.

A system "146.84.28.22" in privateNetworkID 12 has unique name:
12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0/0/0/0/1/0/1/1/0/system.

An endpoint "146.84.28.22" in privateNetworkId 12 has unique name:
12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0/0/0/0/1/1/1/0/0/0/0/0/1/0/1/1/0/endpoint.

By using an IP-address, binary-tree, naming space, one can group all the IP addresses under a subnet in the same naming space that need to be checked by the security service.

For example, one can set up all IP addresses under subnet "146.84.0.0:255.255.0.0" under the naming space 12/1/0/0/1/0/0/1/0/0/1/0/1/0/1/0/0

and set the access rights based on this node name.

With reference now to **Figure 5C**, the IP Object Security Hierarchy is depicted. Under the root, there are two fixed security groups. One is "default" and the other is "all". The name of "default" can be configured by within the configuration service. Users are allowed to configure which subnets are under which customer by using the configuration service.

Under the first level security group, there are router groups and subnet groups. Those systems that have only one interface will be placed under the subnets group. Those systems that have more than one interface will be placed under the router group; a multi-home system will be placed under the router group.

Every IP object has a "securityGroup" field to store which security group it is in. The following describes how security groups are assigned.

When a subnet is created and it is not configured for any customers, its securityGroup is "/default/subnet/*subnetAddress*". When a subnet is created and it is configured in the "customer1" domain, its "securityGroup" value is "/customer1/subnet/*subnetAddress*".

When an IP endpoint is created and it is not configured for any customers, its "securityGroup" value is "/default/subnet/*subnetAddress*". The subnet address is the address of the subnet in which the IP endpoint is located. When an IP endpoint is created and it is configured in the "customer1" domain, its "securityGroup" value is "/customer1/subnet/*subnetAddress*". The subnet address is the address of the subnet in which the IP

endpoint is located.

When a single interface IP system is created, it has the same "securityGroup" value that its interface has. When a router or multi-home system is created, the "securityGroup" value depends on whether all of the interfaces in the router or multi-home system are in the same customer group or not. If all of the interfaces of the router or multi-home system are in the same customer group, e.g., "customer1", its "securityGroup" value is "/customer1/router". If the interfaces of the router or multi-home system are in more than one domain, its "securityGroup" value is "/all/router".

These are the default security groups created by an IP driver. After the security group is created for an object, IP driver will not change the security group unless a customer wants to change it.

The IP Monitor Controller, shown in **Figure 5A**, is responsible for monitoring the changes of IP topology and objects; as such, it is a type of polling engine, which is discussed in more detail further below. An IP driver stores the last polling times of an IP system in memory but not in the IPOP database. The last polling time is used to calculate when the next polling time will be. Since the last polling times are not stored in the IPOP database, when an IP Driver initializes, it has no knowledge about when the last polling times occurred. If polling is configured to occur at a specific time, an IP driver will do polling at the next specific polling time; otherwise, an IP driver will spread out the polling in the polling interval.

The IP Monitor Controller uses SNMP polls to

determine if there have been any configuration changes in an IP system. It also looks for any IP endpoints added to or deleted from an IP system. The IP Monitor Controller also monitors the statuses of IP endpoints in an IP system. In order to reduce network traffic, an IP driver will use SNMP to get the status of all IP endpoints in an IP system in one query unless an SNMP agent is not running on the IP system. Otherwise, an IP driver will use "Ping" instead of SNMP. An IP driver will use "Ping" to get the status of an IP endpoint if it is the only IP endpoint in the system since the response from "Ping" is quicker than SNMP.

With reference now to **Figure 6**, a block diagram shows a set of components that may be used to implement scope-based security access in the present invention. Login security subsystem **602** provides a typical authentication service, which may be used to verify the identity of users during a login process. All-user database **604** provides information about all users in the DKS system, and active user database **606** contains information about users that are currently logged into the DKS system.

Discovery engine **608**, similar to discovery controller **506** in **Figure 5**, detects IP objects within an IP network. Polling engine, similar to monitor controller **516** in **Figure 5**, monitors IP objects. A persistent repository, such as IPOP database **612**, is updated to contain information about the discovered and monitored IP objects. IPOP also obtains the list of all users from the security subsystem which queries its all-users database **604** when initially creating a DSC

(Device Scope Context) object. During subsequent operations to map the location of a user to an ORB, the DSC manager will query the active user database **606**.

The DSC manager queries IPOP for all endpoint data during the initial creation of DSCs and any additional information needed, such as decoding an ORB address to an endpoint in IPOP and back to a DSC using the IPOPOid, the ID of a network object as opposed to an address.

An administrator can fill out the security information with respect to access user or endpoint access and designate which users and endpoints will have a DSC. If not configured by the administrator, a default DSC will be used. While not all endpoints will have an associated DSC, IPOP endpoint data **612**, login security subsystem **602**, and security information **604** are needed in order to create the initial DSCs.

The DSC manager, acting as a DSC data consumer, explained in more detail further below, then listens on this data waiting for new endpoints or users or changes to existing ones. DSC configuration changes are advertised by a responsible network management application, such as a configuration service. Some configuration changes will trigger the creation of more DSCs, while others will cause DSC data in the DSC database to be merely updated.

All DSCs are stored in DSC database **618** by DSC creator **616**, which also fetches DSCs upon configuration changes in order to determine whether or not a DSC already exists. The DSC manager primarily fetches DSCs from DSC database **618**, but also adds runtime information, such as ORB ID.

With reference now to **Figure 7A**, a flowchart depicts a portion of an initialization process in which a network management system prepares a security subsystem for user access to network-related objects. The process begins with the assumption that a network administrator has already performed configuration processes on the network such that configuration information is properly stored where necessary. The discovery engine performs a discovery process to identify IP objects and stores these in the IPOP persistence storage (step **702**).

The DSC creator in the DSC manager generates "initial" DSC objects and stores these within the DSC database (step **704**).

A source user then performs a login on a source endpoint (step **706**). An application may use a resource, termed a target resource, located somewhere within the distributed system, as described above. Hence, the endpoint on which the target resource is located is termed the "target endpoint". The endpoint on which the application is executing is termed the "source endpoint" to distinguish it from the "target endpoint", and the user of the application is termed the "source user".

As part of the login process, the security subsystem updates the active user database for the ORB on which the application is executing (step **708**). The initialization process is then complete.

With reference now to **Figure 7B**, a flowchart depicts further detail of the initialization process in which the DSC objects are initially created and stored. **Figure 7B** provides more detail for step **704** shown in **Figure 7A**.

The process shown in **Figure 7B** provides an outline

for the manner in which the DSC manager sets up associations between users and endpoints and between endpoints and endpoints. These associations are stored as special objects termed "DSC objects". A DSC object is created for all possible combinations of users and endpoints and for all possible combinations of endpoints and endpoints. From one perspective, each DSC object provides guidance on a one-to-one authorization mapping between two points in which a first point (source point) can be a user or an endpoint and a second point (target point) is an endpoint.

Figure 7B depicts the manner in which the DSC manager initially creates and stores the DSC objects for subsequent use. At some later point in time, a user associated with an application executing on a source endpoint may request some type of network management action at a target endpoint, or a network management application may automatically perform an action at a target endpoint on behalf of a user that has logged into a source endpoint. Prior to completing the necessary network management task, the system must check whether the source user has the proper authorization to perform the task at the target endpoint.

Not all network monitoring and management tasks require that a user initiate the task. Some network management applications will perform tasks automatically without a user being logged onto the system and using the network management application. At some point in time, an application executing on a source endpoint may automatically attempt to perform an action at a target endpoint. Prior to completing the necessary network

management task, the system must check whether the source endpoint has the proper authorization to perform the task at the target endpoint in a manner similar to the case of the source user performing an action at a target endpoint.

When the system needs to perform an authorization process, the previously created and stored DSC objects can be used to assist in the authorization process. By storing the DSC objects within a distributed database, a portion of the authorization process has already been completed. Hence, the design of the system has required a tradeoff between time and effort invested during certain system configuration processes and time and effort invested during certain runtime processes. A configuration process may require more time to complete while the DSC objects are created, but runtime authorization processes become much more efficient.

The DSC objects are created and stored within a distributed database during certain configuration processes throughout the system. A new system usually undergoes a significant installation and configuration process. However, during the life of the system, endpoints may be added or deleted, and each addition or deletion generally requires some type of configuration process. Hence, the DSC objects can be created or deleted as needed on an ongoing basis.

The network management framework also provides an additional advantage by storing the DSC objects within a highly distributed database. Because the present invention provides a network management system for an application framework over a highly distributed data

processing system, the system avoids centralized bottlenecks that could occur if the authorization processes had to rely upon a centralized security database or application. The first DSC fetch requires relatively more time than might be required with a centralized subsystem. However, once fetched, a DSC is cached until listeners on the configuration data signal that a change has occurred, at which point the DSC cache must be flushed.

The process in **Figure 7B** begins with the DSC manager fetching endpoint data from the IPOP database (step **710**). The IPOP database was already populated with IP objects during the discovery process, as mentioned in step **702** of **Figure 7A**. The DSC manager fetches user data from the all-user database in the security subsystem (step **712**). Configuration data is also fetched from the configuration service database or databases (step **714**), such as ORB IDs that are subsequently used to fetch the ORB address. A network administration application will also use the configuration service to store information defined by the administrator. The DSC manager then creates DSC objects for each user/endpoint combination (step **716**) and for each endpoint/endpoint combination (step **718**), and the DSC object creation process is then complete.

With reference now to **Figure 7C**, a flowchart depicts further detail of the initial DSC object creation process in which DSC objects are created and stored for an endpoint/user combination. **Figure 7C** provides more detail for step **716** in **Figure 7B**. The process shown in **Figure 7C** is a loop through all users that can be

identified within the all-user database. In other words, a set of user accounts or identities have already been created and stored over time. However, all users that have been authorized to use the system do not have the same authorized privileges. The process shown in **Figure 7C** is one of the first steps towards storing information that will allow the system to differentiate between users so that it can adaptively monitor the system based partially on the identity of the user for which the system is performing a monitoring task.

The process in **Figure 7C** begins by reading scope data for a target endpoint from the IPOP database (step **720**). The DSC creator within the DSC manager then reads scope data for a source user from the IPOP database (step **722**). A determination is then made as to whether or not the source user is allowed to access the target endpoint (step **724**). This determination can be made in the following manner. After the initial DSC is obtained, the source user information is used to make an authorization call to the security subsystem as to whether or not the source user has access to the security group defined in the DSC. It may be assumed that the security system can perform this function efficiently, although the present invention does not depend on auto-generation of security names or security trees. The present invention should not be understood as depending upon any particular implementation of security authorization.

If not, then the process branches to check whether another user identity should be processed. If the source user is allowed to access the target endpoint, then a DSC object is created for the current source user and current

target endpoint that are being processed (step **726**). The DSC object is then stored within the DSC database (step **728**), and a check is made as to whether or not another source user identity requires processing (step **729**). If so, then the process loops back to get and process another user, otherwise the process is complete.

With reference now to **Figure 7D**, a flowchart depicts further detail of the initial DSC object creation process in which DSC objects are created and stored for an endpoint/endpoint combination. **Figure 7D** provides more detail for step **718** in **Figure 7B**. The process shown in **Figure 7D** is a loop through all endpoints that can be identified within the IPOP database; the IPOP database was already populated with IP objects during the discovery process, as mentioned in step **702** of **Figure 7A**. During runtime operations, an application executing on a source endpoint may attempt to perform an action at a target endpoint. However, not all endpoints within the system have access to requesting actions at all other endpoints within the system. The network management system needs to attempt to determine whether or not a source endpoint is authorized to request an action from a target endpoint. The process shown in **Figure 7D** is one of the first steps towards storing information that will allow the system to differentiate between endpoints so that it can monitor the system based partially on the identity of the source endpoint for which the system is performing a monitoring task.

The process in **Figure 7D** begins by reading scope data for a target endpoint from the IPOP database (step

730). The DSC creator within the DSC manager then reads scope data for a source endpoint from the IPOP database (step **732**). A determination is then made as to whether or not the source endpoint is allowed to access the target endpoint (step **734**) based on the scope defined in the DSC. For example, a simple scope of X.Y.Z.* will allow an address of X.Y.Z.Q access. If not, then the process branches to check whether another source endpoint should be processed. If the source endpoint is allowed to access the target endpoint, then a DSC object is created for the source endpoint and target endpoint that are currently being processed (step **736**). The DSC object is then stored within the DSC database (step **738**), and a check is made as to whether or not another source endpoint requires processing (step **739**). If so, then the process loops back to get and process another endpoint, otherwise the process is complete.

The components and subsystems described above with respect to **Figures 6-7D** are used to restrict administrative user access to various operations and functionality within the network management framework. The following description of **Figures 8A-14B** describe particular operations and functionality that may be available to administrative users within the network management framework. In particular, **Figures 8A-14B** show various ways in which scopes and anchors are used to perform certain tasks within a single network management framework that is being used to manage a multi-customer environment. More specifically, **Figures 8A-9B** depict a manner in which an administrative user may view the

topology of a set of network-related objects that are being managed by the network management framework, whereas **Figures 10A-14B** depict a process for configuring information within the network management framework so that it may manage the networks of multiple customers.

As described above, an IP driver subsystem is implemented as a collection of software components for discovering, i.e. detecting, network "objects", such as IP networks, IP systems, and IP endpoints by using physical network connections. The collected data is then provided through other services via topology maps accessible through a GUI or for the manipulation of other applications. The IP driver system can also monitor objects for changes in IP topology and update databases with the new topology information. The IPOP service provides services for other applications to access the IP object database.

Referring again to **Figure 5A**, IP driver subsystem **500** contains a conglomeration of components, including one or more IP drivers **502**. Every IP driver manages its own "scope", and every IP driver is assigned to a topology manager within topology service **504**, which stores topology information obtained from discovery controller **506**. The information stored within the topology service may include graphs, arcs, and the relationships between nodes determined by IP mapper **508**. Users can be provided with a GUI to navigate the topology, which can be stored within a database within the topology service.

The topology service provides a framework for DKS-enabled applications to manage topology data. In a

manner similar to the IPOP service, the topology service is actually a cluster of topology servers distributed throughout the network. All of the functions of the topology service are replicated in each topology server. Therefore, a client can attach to any server instance and perform the same tasks and access the same objects. Each topology-related database is accessible from more than one topology server, which enables the topology service to recover from a server crash and provide a way to balance the load on the service.

Topology clients create an instance of a TopoClientService class. As part of creating the TopoClientService instance, the class connects to one of the topology servers. The topology server assumes the burden of consolidating all of the topology information distributed over the different topology servers into a single combined view. The topology service tracks changes in the objects of interest for each client and notifies a client if any of the objects change.

The topology service may have a server-cluster design for maximizing availability. As long as there is at least one instance of the topology server running, then clients have access to topology objects and services. The topology service design allows for servers to occasionally fail. Each server is aware of the state of all the other server instances. If one instance fails, the other servers know immediately and automatically begin to rebuild state information that was lost by the failed server. A client's TopoClientService instance also knows of the failure of the server to which it is connected and re-connects to a different server.

The objects residing at a failed topology server are migrated to the other topology servers when the drivers owning those objects have re-located.

The topology service is scalable, which is important so that the service may be the central place for all network topology objects for all of the different DKS-related applications in order to provide efficient service for millions of objects. As the number of clients, drivers, and objects increase, an administrator can create more instances of topology servers, thereby balancing the workload. Using the server cluster approach, any growth in the number of clients, drivers, and objects is accommodated by simply adding more servers. The existing servers detect the additional instances and begin to move clients and drivers over to the new instances. The automated load-balancing is achieved because the clients and objects are not dependent on any one server instance.

In order to provide a service for an entire enterprise, all of the enterprise's objects generally do not reside in the same database. There may be many reasons that make it undesirable to require that all topology objects be stored in the same database instance. For example, a database simply may not be reachable across an international boundary, or the volume of information going into the database may exceed a single database's capacity. Therefore, the topology objects may span databases, and there may be relationships between objects in different databases. However, it may be assumed that all topology objects in a domain reside in the same database. For example, all IP objects for a

single enterprise do not necessarily reside in the same database as the enterprise's IP space may be split into many domains, e.g., a southwest IP domain and a northeast IP domain, but each domain may reside in different databases and still have relations between their objects. Hence, it is possible to have two objects related to each other even though they are in different databases. Since the name of the domain is part of the id of the object, each object can be uniquely identified within the entire topology service.

When an application is installed and configured to use the DKS services, the application provides some information to the topology service about the different types of TopoObjects it will be creating. This class information closely resembles the network entities that a driver will be managing. For example, an IP application works with Network, System, and Endpoint resource types, as described previously with respect to **Figure 4**. Giving TopoObjects a resource type enables client applications to identify, group, and query the databases based on domain-specific types. Each resource type may have many different types of relations that the driver may create, and the most common type may be the containment relation, which shows the containment hierarchy of a domain. Each relation type has a corresponding ViewData object, which provides information that an administrative console needs to create a view of the TopoObjects. For example, the ViewData object may contain members like BackgroundColor and LayoutType that are used to construct a graphical display of the object. Relations can be created between any two TopoObjects. The TopoObjects can be owned by the

same driver, different drivers in the domain, or even drivers in different domains.

With reference now to **Figure 8A**, a flowchart depicts a process for creating topology data. The process begins when one or more discovery engines scan physical networks until a new device is found (step **802**). A determination is made as to whether or not a network object exists for the network in which the endpoint has been found (step **804**). If not, then a network object is created (step **806**), otherwise the process continues.

In either case, a determination is then made as to whether or not a system object exists for the system in which the endpoint has been found (step **808**). If not, then a system object is created (step **810**), otherwise the process continues. In either case, an endpoint object is then created for the discovered device (step **812**), and all of the created objects are then stored within the IPOP database (step **814**). The created objects are then mapped into the current topology (step **816**), and the topology service creates topology objects (step **818**) and stores them within the topology database (step **820**). The process of discovering a physical network or device and storing appropriate information is then complete.

With reference now to **Figure 8B**, a flowchart depicts a process for listening for physical network changes that affect topology objects. The process begins with a determination of whether or not one or more polling engines has found a system or device that has failed (step **832**). If not, then a determination is made as to whether or not a new device has been discovered (step

834). If not, then the process loops back to continue monitoring the networks.

If either a new device is discovered or a device has failed, then the appropriate changes are made to the objects representing the physical devices that have been affected by updating the IPOP database (step **836**). For example, if a new device is found, then appropriate steps are made to create the necessary objects in a manner similar to steps **804-820** in **Figure 8A**. A determination is then made as to whether or not the detected change affects the topology (step **838**), and if not, then the process is complete. If the topology has been affected, then the topology database is updated as necessary (step **840**), and the process of listening for network changes and reflecting those changes within the topology is complete.

With reference now to **Figure 9A**, a figure depicts a graphical user interface window that may be used by a network or system administrator to view the topology of a network that is being monitored. Window **900** depicts a simple network showing router device **902**, endpoint **904**, and endpoint **906**. In addition, line **908** shows a relation between endpoint **904** and router **902**, and line **910** shows a relation between endpoint **906** and router **902**. Each of the icons **902-906** represents a TopoObject that is maintained by the topology service.

With reference now to **Figure 9B**, a figure depicts a graphical user interface window that shows the topology of a network that has changed. Window **930** in **Figure 9B** shows the same network as depicted within window **900** of

Figure 9A except that an endpoint has failed and has been deleted from the current topology. Window **930** depicts a simple network showing router device **932**, endpoint **934**, and line **936** for the relation between endpoint **934** and router **932**.

As noted previously, **Figures 8A-9B** depict one administrative operation that may be performed in conjunction with the present invention, whereas **Figures 10A-14B** depict another administrative operation; these examples are included herein to provide a context in which the network management framework uses IP drivers, scopes, anchors, and other framework components in order to provide a robust network management framework for supporting the management of the networks of multiple customers in an integrated manner.

One particular problem that a robust network management framework for a service provider must confront is the fact that many customers of a service provider may have software-based and/or hardware-based network address translators, or NATs. Each network address within a given domain serviced by a NAT can be assumed to be unique. However, across multiple NATs, each network address within the entire set of network addresses cannot be assumed to be unique. In fact, the potential for duplicate addresses over such a large, highly distributed network is quite high. Even if the service provider is managing only one customer within a particular network management environment, the same problem might also exist because a single customer may operate multiple NATs for multiple networks. This type of problem is illustrated in more detail in **Figures 10A-10B**.

With reference now to **Figure 10A**, a block diagram depicts a known configuration of software and/or hardware network components linking multiple networks. A computer-type device is functioning as firewall/NAT **1020**, which is usually some combination of software and hardware, to monitor data traffic from external network **1022** to internal protected network **1024**. Firewall **1020** reads data received by network interface card (NIC) **1026** and determines whether the data should be allowed to proceed onto the internal network. If so, then firewall **1020** relays the data through NIC **1028**. The firewall can perform similar processes for outbound data to prevent certain types of data traffic from being transmitted, such as HTTP (Hypertext Transport Protocol) Requests to certain domains.

More importantly for this context, the firewall can prevent certain types of network traffic from reaching devices that reside on the internal protected network. For example, the firewall can examine the frame types or other information of the received data packets to stop certain types of information that has been previously determined to be harmful, such as virus probes, broadcast data, pings, etc. As an additional example, entities that are outside of the internal network and lack the proper authorization may attempt to discover, through various methods, the topology of the internal network and the types of resources that are available on the internal network in order to plan electronic attacks on the network. Firewalls can prevent these types of discovery practices.

While firewalls may prevent certain entities from

obtaining information from the protected internal network, firewalls may also present a barrier to the operation of legitimate, useful processes. For example, in order to ensure a predetermined level of service, benevolent processes may need to operate on both the external network and the protected internal network; a customer system is more efficiently managed if the management software can dynamically detect and dynamically configure hardware resources as they are installed, rebooted, etc. Various types of discovery processes, status polling, status gathering, etc., may be used to get information about the customer's large, dynamic, distributed processing system. This information is then used to ensure that quality-of-service guarantees to the customer are being fulfilled. However, firewalls might block these system processes, especially discovery processes.

Firewall/NAT **1020** also performs network address translation between addresses on external network **1022** and addresses on internal network **1024**. In the example, system **1030** connects to internal network **1024** via NIC **1032**; system **1034** connects to internal network **1024** via NIC **1036**; and system **1038** connects to internal network **1024** via NIC **1040**. Each NIC has its own MAC (Media Access Control layer) address, which is a guaranteed unique hardware address in the NIC that is used to address data packets to and from the system that is using a given NIC. Network Address Translator (NAT) **1020** presents all of the systems on internal network **1024** to external network **1022** with a single, public, IP address.

However, systems **1030**, **1034**, and **1038** have addresses which are unique within internal network **1024**. NAT **1020** retrieves the addresses within the data packets flowing between the internal network and the external network, translates the addresses between the two domains, stores the addresses back into the data packets, and forwards the data packets.

The internal network supports a private address space with globally non-unique address, whereas the external network represents a public address space of globally unique addresses. A NAT device is used to manage the connectivity of the private network with the outside world; the NAT device bridges the internal network and the external network and converts addresses between the two address spaces. Within a private network behind a NAT, an enterprise may have its own private address space without concern for integrating the private address space with the global Internet, particularly with the predominant IPv4 address space that is currently in use.

NATs are helpful for certain enterprises that do not require full connectivity for all of its devices. However, NATs present barriers for certain functionality. A NAT must have high performance in order to perform address translation on all data packets that are sent and received by a private network. In addition, a network management framework for a highly distributed system may be forced to coordinate its actions across multiple NAT devices within a single customer or across multiple customers. For example, systems **1030**, **1034**, and **1038** have addresses which are unique within internal network

1024. However, another internal network within the same enterprise may duplicate the addresses that are used within internal network **1024**.

When contending with multiple NATs, the network management framework cannot assume uniqueness among private network addresses. In some prior art systems, it would have been straightforward to use the private network address of a device as a unique key within the network management applications because the private network address has a unique association with a networked device. In a highly distributed system, the network management framework needs to store many data items in an efficient manner yet cannot rely upon a scheme that uses the private network addresses as unique keys for managing those data items.

Future IT solutions may not need to confront the same problems because the Internet is moving towards using a new standard IP protocol known as IP Version 6 (IPv6) that will have a much larger address space. However, a current network management solution must confront legacy issues of maintaining currently installed hardware.

Prior art solutions have generally included dedicated boxes or devices that perform address translation. These solutions tend to be specific modifications to an operating system or kernel, which reduces the benefit of having standardized implementations of software platforms. In other words, some applications may not be compatible with the solution. In addition, such solutions may require installing a dedicated device for each system, which is

prohibitive.

With reference now to **Figure 10B**, a block diagram depicts a service provider connected to two customers that each have subnets that may contain duplicate network addresses. As noted above, multiple internal networks within a highly distributed data processing system may contain duplicate addresses. Service provider **1050** manages networks and applications for multiple customers and stores its data within multi-customer database **1052**. Customer **1054** has a network of devices that includes subnet **1056** that connect with the larger network through NAT **1058**; customer **1064** has a network of devices that includes subnet **1066** that connect with the larger network through NAT **1068**. Duplicate network addresses could appear within subnets **1056** and **1066**. In order to provide certain services in a seamless fashion such that both customers can be managed by the service provider as a single logical network, the service provider requires a network management framework that can handle duplicate network addresses.

The network management framework that is described herein can manage multiple networks over which duplicate addresses might appear, e.g., as shown in **Figure 10B**, such that the distributed network management framework is operable across multiple NATs. The manner in which the network management is performed is described further below in more detail.

With reference now to **Figure 11A**, a block diagram shows a set of components that may be used to implement multi-customer management across multiple networks in

which duplicate addresses may be present. **Figure 11A** depicts components that are similar to components introduced in other figures above. User security subsystem **1106** provides a user authentication and authorization service, which may be used to verify the identity of users, such as administrators, during a login process and during administrative operations. IP drivers **1108** detect IP objects within an IP network. Gateway/NEL service **1110** provides action object processing within gateways. A persistent repository, such as IPOP database **1112**, is updated to contain information about the discovered and monitored IP objects. Other ORB or core services **1114** may also access IPOP database **1112**.

Customer address manager service **1116** queries IPOP **1112** during operations that allow an administrator to resolve addressability problems. Customer logical network creator **1118** fetches administrator input about the groupings of physical networks into a logical network, as may be provided by an administrator through an application GUI, such as the GUI shown in **Figure 13**. From this input, the various scopes of the physical networks are combined to create a logical scope as previously described above.

VPN creator **1120** fetches administrator input concerning which physical networks belong to which customer. The administrator can provide a name to each physical network collection, which is used by the anchorname creator **1122** to define an anchorname, which is highest level name used to describe a logical network. The final name of each physical network is a combination

of the anchorname and the name assigned to each logical network. For example, the name of a logical network consisting of the physical networks 146.5.*.* would be "austin\downtown\secondfloor" comprising the anchorname="austin" and the name="downtown\secondfloor." The anchorname creator supplies a name to the IP Driver subsystem by combining the anchorname, determined from the scope configuration, and the name of the physical network element object from IPOP. Finally, a customer ID creator **1124** uses the collection of IDs used by all customers to generate a new unique customer identifier when required by IPOP; the identifiers are used rather than strings for efficient database searches of large number of network objects. During subsequent operations to map the location of a user to an ORB, customer address manager service **1116** queries may query an active-user database similar to that shown in **Figure 6.**

With reference now to **Figures 11B-11D**, some simplified pseudo-code depicts the manner in which action objects and endpoint objects can be implemented in an object-oriented manner. **Figure 11B** shows a class for action objects, while **Figures 11C-11D** show classes for endpoint objects.

With reference now to **Figure 12A**, a flowchart depicts a portion of an initialization process in which a network management system prepares for managing a set of networks with multiple NATs. It is assumed that a network administrator has already performed configuration processes on the network such that configuration information is properly stored where necessary. The process begins when a multi-customer administrator

creates DKS VPN IDs during installation (step **1202**). For example, after the ORB has started, the ORB starts a Command Line Interface (CLI) Service through which an administrator can issue CLI commands to create VPN IDs within the IPOP database, such as "ipop create VPN" used by customer and VPN ID creator **1124** in order to create a unique customer ID or a unique VPN ID.

The process then continues with the multi-customer administrator creating network scope for one or more customers (step **1204**). Multi-customer regions may also be created, which refers to the managing a region that consists of two or more customers for which care has to be taken not to intermix different customer data. At this point, the physical network is discovered via a discovery engine, such as the IP driver service, which performs a discovery process to identify IP objects and stored those in the IPOP persistence storage. For all customer locations, all of the physical networks that have been discovered are displayed to the administrator so that the administrator can conveniently apply names to the discovered objects/networks. In addition, multiple address problems are determined and displayed to the administrator, who is then required to assign a VPN ID to a customer, e.g., by using an application GUI such as that shown in **Figure 13**.

Part of the customer scope, i.e. a logical scope consisting of a collection of physical networks as described previously, is the customer anchorname text array, e.g., "ibm\usa\Austin", the customer name, and a unique customer ID, as created by customer address manager service **1116** in **Figure 11A**. The hash number is

computed by the customer base text name, e.g., "ibm", the network addresses, e.g., all subnets of reserved public addresses, and VPN IDs.

The administrator then resolves any outstanding addressability problems (step **1206**). For example, a large corporation may have subnets "10.0.0.*" on each floor of a large office. After those have been resolved, then the system stores the mapping of customers, VPN IDs, customer anchornames, and customer networks in the IPOP database (step **1208**), and the initialization process is then complete.

With reference now to **Figure 12B**, a flowchart depicts further detail of the initialization process in which the administrator resolves addressability problems. **Figure 12B** provides more detail for step **1206** shown in **Figure 12A** in which an administrator proceeds to resolving identified addressability problems.

The process begins, during the initialization process, as an ORB starts the customer address manager (step **1212**). The customer address manager then finds the identity of the administrator that is performing various address management functions through a network management application (step **1214**). At this point, the identity of the network administrator may be used to ensure than the administrator has the proper authorization parameters. However, for the sake of explanation, it may be assumed that an administrator with multi-customer rights has access to the GUI to create VPNs for multiple customers, i.e. it may be assumed that an administrator has the proper authorization for working with the data from multiple customers. The multi-customer administrator

uses the administrator GUI shown in **Figure 13**, which uses the customer address manager, to display all of the discovered networks for the administrator's customer or customers (step **1216**).

After retrieving this information, the customer address manager may then allow the administrator to assign VPN IDs to those networks for which it can be determined that the networks have an addressability problem (step **1218**). The assigned VPN IDs are then stored as updated information within the network objects within IPOP (step **1220**). The scope information is also updated with a VPN ID (step **1222**); initially, many scopes are defined as "VPN = 0", which means no VPN address. The VPN ID creator ensures that unique VPN IDs are created such that duplicate addresses can exist within a VPN that has an assigned VPN ID. This portion of the initialization process is then complete. The manner in which the administrator assigns VPN IDs is explained in more detail with respect to **Figure 12C**.

In order to determine which networks require a VPN ID, the customer address manager sorts through all of the network addresses, looking for problematic addresses. For example, a set of 255 public addresses, such as "10.0.0.*", are reserved for local network purposes. Hence, even if two networks within the network management system do not have colliding local network addresses, the potential for future collisions exists.

With reference now to **Figure 12C**, a flowchart depicts further detail of the process in which the administrator assigns VPN IDs. **Figure 12C** provides more detail for step **1218** shown in **Figure 12B**. The process

begins by displaying those networks have been determined to need a VPN ID assigned since a duplicate address exists, as determined with respect to step **1218** above, to the current administrator (step **1232**). The customer address manager then displays a list of possible VPN IDs from which the administrator may choose (step **1234**), and the administrator is able to define VPN IDs as necessary if not already defined (step **1236**). VPN IDs could have been previously defined through the configuration service, most likely during installation. However, at configuration time, the networks have not yet been discovered, so it is not possible for the system to know if and where duplicate addresses exist. While the figures are described with respect to the actions of a single administrator, a highly distributed system has a collection of administrators that are typically not in one location. Hence, one of goals of the DKS management framework is to detect errors and allow the administrators to have input into the manner in which the errors should be corrected.

A determination is then made as to whether the administrator is a multi-customer administrator (step **1238**). If not, then the VPN ID that has been chosen by the administrator can be assigned to the networks of the administrator's customer (step **1240**). If the administrator is a multi-customer administrator, then the customer address manager must get a specific customer from the administrator (step **1242**), and the chosen VPN ID is assigned to the specified customer (step **1244**). This portion of the initialization process is then complete. After these initialization steps, the administrator has

an overall addressing scheme that should be coherent. The IP addresses, VPN IDs, and other information, when taken together, provides a scheme for unique identifiers that the management system can use to manage the devices throughout the system.

With reference now to **Figure 13**, a figure depicts a graphical user interface window that may be used by a network or system administrator to set monitoring parameters for resolving address collisions. Window **1350** is a dialog box that is associated with a network management application; a system or network administrator is required to create or enter VPN IDs to resolve duplicate addresses that have been detected, such as physical network addresses **1352** and **1354**. An administrator could also invoke the application on a regular basis when necessary, or it could be invoked automatically by the network management system when address collisions are detected. "Set" button **1374** and "Clear" button **1376** allow the administrator to store the specified values or to clear the specified parameters. Checkbox **1378** allows an administrator to quickly change the VPN ID for an entire physical scope indicated within window **1350**.

Figures 14A-14B depict examples of processes that may be performed by the network management system after system configuration/initialization when an administrator is using a network management application to perform a certain operation, such as a simple IP "Ping" command as shown in the example. While the example shows a simple IP "Ping" action, a more complex action could include a

software distribution application that installs software on endpoints throughout the distributed data processing system.

With reference now to **Figure 14A**, a flowchart shows the overall process for performing an IP "Ping" within a multi-customer, distributed data processing system containing multiple private networks. The process begins when an ORB starts a private network multi-customer manager (PNMCM) that is used by the system to perform certain actions, such as requesting an IP "Ping" (step **1402**). The user of the application, which in this case is a network or system administrator for a particular customer, launches an application associated with the PNMCM (step **1404**). Within the application, the administrator chooses an endpoint and requests an IP "Ping" action, most likely from hitting a "Ping" button within the GUI (step **1406**).

The PNMCM manager attempts to fetch the requested endpoint from the IPOP database using only the IP address as specified or selected by an administrator within an application GUI (step **1408**). A determination is then made as to whether IPOP returns duplicate endpoints (step **1410**). If not, then the process branches to show the results of the requested "Ping" action.

If there is a collision among duplicate IP addresses, they are displayed to the administrator along with the previously associated VPN IDs that help to uniquely identify the endpoints (step **1412**). The administrator is requested to choose only one of the duplicate endpoints (step **1414**), and after choosing one, the administrator may request to perform the "Ping"

action on the selected endpoint (step **1416**). The PNMCM displays the results of the "Ping" action to the administrator (step **1418**), and the process is complete.

With respect to **Figure 14B**, a flowchart depicts a process for obtaining and using an application action object (AAO) within the network management system of the present invention. An application action object is a class of objects that extends an action object class in a manner that is appropriate for a particular application.

The process begins when an application requests, from the gateway service, an application action object (AAO) for a "Ping" action (AAOIP) against a target endpoint (step **1422**). The process assumes that the administrator has already chosen the source and target endpoints through some type GUI within a network management application.

The gateway service asks the NEL service to decode the target endpoint from the request (step **1424**). As noted previously, one of the primary roles of the NEL service is to support the requests from applications for routes, as explained above with respect to **Figure 3**. The NEL service then asks the IPOP service to decode the endpoint object (step **1426**). Assuming that the processing has been successfully accomplished, IPOP returns an appropriate AAOIP object to the NEL service, including VPN ID if necessary (step **1428**), and the NEL service returns the AAOIP object to the gateway service (step **1430**). The gateway service then returns the AAOIP object to the application (step **1432**). The application then performs the desired action (step **1434**), such as an

IP "Ping", and the process is complete.

The description of **Figures 11A-14B** explain a methodology for configuring the network management framework to manage the networks of multiple customers, which may have conflicting physical network addresses; by employing scopes and anchors, the conflicts can be resolved. As described above with respect to **Figures 5A-5C**, when a network is discovered, it is associated with a particular scope, and each scope relates to an anchor. Scope configuration is important to the proper operation of the IP drivers because IP drivers assume that there are no overlaps in the drivers' scopes. Since there should be no overlaps, every IP driver has complete control over the objects within its scope. When IP mapper is informed that a new network has been discovered, it uses the scope associated with the network to determine with which anchor, i.e. customer, the newly discovered network should be associated, and the appropriate updates are also made within the topology database.

Assuming that an administrative user has the proper security access, the administrative user should have the ability to modify various management aspects of scopes and anchors. However, scope manipulation can be quite complex. For example, an administrative user may incorrectly attempt to change the manner in which a set of endpoints is managed such that a conflict between scopes arises. Hence, the network management framework needs to be able to handle dynamic changes in scope while monitoring for errors such as scope overlap. In addition, the network management framework should enable

dynamic changes in administrative responsibility over different scopes and also enable security monitoring for such administrative changes. These types of changes are described in a high-level manner with respect to **Figure 15**.

With reference now to **Figure 15**, a block diagram depicts a logical organization of network-related objects that may be managed within different scopes on behalf of a plurality of customers whose physical networks are being administered using the network management framework of the present invention. After a discovery process, a distributed data processing system is determined to contain physical devices that may be represented by the network-related objects shown in **Figure 15**: endpoints **1501-1506** have been dynamically discovered; endpoint **1504** and endpoint **1505** reside within system **1507**; endpoint **1501**, endpoint **1502**, and endpoint **1504** reside within network **1508**, while endpoint **1503**, endpoint **1505**, and endpoint **1506** reside within network **1509**. In this example, system **1507** may be a router.

At some point in time during the initial phases of configuring a distributed data processing system, the physical networks of the distributed data processing system may have been discovered, but the physical networks would not yet have been associated or assigned to logical networks on behalf of customers. In this case, all of the network-related objects **1501-1509** are managed using a single IP driver within a single scope; for instance, this initial scope may be initially assigned to a management entity, i.e. the service

provider that is managing the distributed data processing system on behalf of a set of customers.

At some subsequent point in time, an administrative user may desire to dynamically segment the distributed data processing system into two customer scopes; the scope of the initial IP driver is changed to include only the endpoints within network **1508**, and another IP driver is introduced to be responsible for monitoring the endpoints within network **1509**. It is assumed that the administrative user has appropriate security access to an administrative application that provides access to a Distributed IP Driver Change Manager (DICM) for initiating such management actions; the DICM may be a component within the IP drivers or may be a separate service that coordinates these types of management actions with the IP drivers.

For this example, the DICM would perform the following deletion actions: endpoint **1503**, endpoint **1505**, and endpoint **1506** would be deleted from the endpoint table is associated with the initial IP driver in the IPOP database; references to endpoint **1505** would be deleted from the system table that is associated with system **1507** in the IPOP database; and references to endpoint **1505** would be deleted from the network table that is associated with network **1509** in the IPOP database.

Although not shown in **Figure 4**, other tables may be stored within the IPOP database for use in these operations to change the configuration of the IP drivers. In this case, the DICM moves the network-related objects

to temporary tables: the system object for system **1507** is moved to an orphan system table in the IPOP database; the network object for network **1509** is moved to an orphan network table in the IPOP database; and the endpoint objects for endpoint **1503**, endpoint **1505**, and endpoint **1506** are moved to an orphan endpoint table in the IPOP database. All of the objects within the orphan object tables are then marked as seeds to be used by the newly introduced IP driver upon its startup to perform an abbreviated, more efficient, discovery process. Rather than configuring the new IP driver instance to physically discover its assigned network of devices, the new IP driver instance can be configured to perform discovery using the objects in IPOP that have been marked as IPOP seeds. When the new IP driver is started on its ORB for the first time, the seeds are used to create new IPOP objects and topology objects, including the relations between objects, with new object IDs and other non-physical network-related data. After that point in time, any management operations that need to be performed with the objects through the new IP driver would require appropriate security access as configured for the new IP driver.

At this point in time, the distributed data processing system can be viewed as two logical networks that are being managed by two IP drivers on behalf of a single management entity. However, an administrative user may then assign each of the logical networks to a customer such that the logical networks are managed on behalf of two customers. From that point on, any administrator-initiated operations would require the

appropriate security access as configured for those customers. Any attempts to change the scopes of the logical networks or to create a new IP driver across customer lines would generate a security event for review by an administrative user, which is explained in more detail with respect to **Figure 16**.

With reference now to **Figure 16**, a flowchart depicts a process for handling certain types of scope-related events. The process begins with the Distributed IP Driver Change Manager (DICM) detecting a configuration change that affects the configuration of the current scopes or introduces a new IP driver (step **1602**). The DICM may register multiple object listeners throughout the distributed data processing system for these purposes.

In response to receiving an event, the DICM determines whether a change has been detected in an anchor name object (step **1604**). If so, then the DICM generates a customer security breach event (step **1606**), which causes a network console application to log the event (step **1608**) and display the event within a GUI for review by an administrative user (step **1610**), after which the process is considered complete. Because a change in anchor name is equivalent to moving a logical network between customers, the change is critical and needs to be verified. The administrative user can then determine whether the event was actually generated by an administrator-initiated action. In this manner, administrative operations are subject to a feedback loop that automatically detects certain changes that affect the customer to which a network-related object is

assigned. In a multi-customer environment with many administrative users, the administrative user that approves of the change may be in a completely different management office than the administrative user that initiated the event.

If the event is not a change in the anchor name, then a more detailed test is performed to determine whether the change has resulted in a change in scope that affects the customer to which a network-related object is assigned. A determination is made as to which IP drivers are affected by the detected change (step **1612**), and for each scope assigned to the determined IP drivers, all network-related objects (networks, systems, and endpoints) are retrieved (step **1614**). The DICM then loops through the retrieved objects by getting a next object (step **1616**) and determining whether the customer associated with the object differs from the other objects within the same scope (step **1618**). If so, there has been some type of breach in security related to the various customers because a single scope can only include objects assigned to a single customer. Hence, the DICM again generates a customer security breach event at step **1606**, which is handled again at steps **1610** and **1612**. Otherwise, if there are more objects to be processed (step **1620**), then the DICM continues to loop through those objects to determine whether there has been a customer security breach event.

With reference now to **Figure 17A-17B**, a pair of flowcharts depicts a process for relocating network-related objects based on a change of scope initiated by an administrator or for configuring multiple

customers within a distributed data processing system. In a manner similar to that described with respect to **Figure 15**, a multicustomer administrator wants to segment a single distributed data processing system into a multi-customer environment. An administrative user changes the scope configuration pertaining to scope or IP driver by defining a new scope to encompass a certain physical network, after which the topology service will be able to display a network object or icon for the newly introduced customer. The change may be effected through a network management console application at the direction of the administrative user.

The process begins by detecting that a new IP driver has been started or that a change has been made in the configuration of one of the existing scopes (step **1702**). All IP drivers that are affected by the change are determined (step **1704**), and the process then loops through the scopes of the affected IP drivers.

A first or next affected IP driver is identified (step **1706**), depending upon the iteration in the loop through the IP drivers, and any objects within one or more scopes of the affected IP driver are examined to determine which objects are now out-of-scope for the affected IP driver, i.e. outside of the scopes assigned to the affected IP driver, based on the newly defined IP driver's scope(s) or the change in the configuration of one or more scopes (step **1708**).

A first or next out-of-scope object is obtained (step **1710**), depending upon the iteration in the loop through the out-of-scope objects, and the out-of-scope object is marked as a seed object (step **1712**), which will

cause the object to be subsequently processed in a special manner when the new IP driver performs its rediscovery process. The out-of-scope object is then moved to an orphan object table in the IPOP database (step **1714**) corresponding to its type of network-related object, i.e. network object, system object, or endpoint object. A determination is then made as to whether there are additional out-of-scope objects to be processed (step **1716**). If so, then the process branches back to step **1710** to process another out-of-scope object. If all of the out-of-scope objects have been processed, then the rediscovery process may begin.

Appropriate configuration values are set for the IP driver that is responsible for the newly configured scope to indicate that the IP driver is to perform a rediscovery process on the seed objects in the IPOP database (step **1718**). A determination is then made as to whether the IP driver is already executing or whether it is being initialized (step **1720**).

If the IP driver was already executing, then the processing within the IP driver differs as it performs certain actions in anticipation of the rediscovery process. As a first step, the configuration listeners for the IP driver detect a change in the IP driver's configuration (step **1722**), most likely in conjunction with notification from a configuration service. In response, the IP driver flushes any write operations that are pending for the IPOP database (step **1724**) to ensure that those updates are reflected in the IPOP database before the IP driver begins performing its monitoring and discovery duties under a new configuration. For similar

reasons, the IP driver flushes its cache of IPOP objects (step **1726**). The IP driver then reads all objects defined to be within its scope (which may be a newly configured scope) from the IPOP database (step **1728**), and for any of those objects that are marked as seed objects, the IP driver performs a rediscovery process in which it creates new IPOP objects for the seed objects (step **1730**). In the rediscovery process, the IP driver regenerates non-physical network information using newly generated identifiers, such as unique numbers that are used by the DKS services, object IDs, anchor names that may have changed for the scope, and relationships between with other objects. The process then branches as the processing of this IP driver within the IP driver loop is complete.

If the IP driver was not already executing at step **1720**, then the IP driver is newly initialized, which it may determine by reading a configuration value that directs the IP driver to perform a rediscovery process using IPOP seeds rather than performing a full discovery process on the networks within its scopes. The rediscovery process for the newly initialized IP driver is similar to that described above. The IP driver reads all objects defined to be within its scope from the IPOP database (step **1730**), and for any of those objects that are marked as seed objects, the IP driver performs a rediscovery process in which it creates new IPOP objects for the seed objects (step **1732**). The process then branches as the processing of this IP driver within the IP driver loop is complete.

A determination is then made as to whether there are

other affected IP drivers (step **1736**), and if so, then the process branches back to step **1706** to process additional scopes. If there are no other affected IP drivers, then additional configuration processing can be performed as part of the operation of the network management console application that initiated the change in scope. For example, if an administrative user was defining a new logical network within the distributed data processing system, then the anchorname that was entered by the user, such as through an application window similar to that shown in **Figure 13**, is associated with all of the objects in a newly defined or newly configured logical network (step **1738**). In addition, all of the affected objects in the IPOP database could be indicated as being "customer-defined" (step **1740**). This indication assists in determining whether there has been an unwarranted modification of an anchor name or customer association as described above with respect to **Figure 16**.

In addition, the IP mapper creates a customer container object for each defined or reconfigured customer (step **1742**) so that the topology service can display the customer objects (step **1744**), and the administrator is notified that the operation is complete (step **1746**).

As noted previously, **Figures 8A-14B** depict a couple of administrative operations that may be performed within the network management framework; these examples are included to provide a context for showing the manner in which the components within the DKS network management framework operate. **Figures 15-17B** depict a methodology

for allowing an administrator for a multi-customer service provider to configure portions of the network management framework for operating on the networks of more than one customer. While **Figures 15-17B** are helpful for showing a particular aspect of the manner in which the DKS network management framework is configured, it does not describe in detail the methodology by which the network management framework can be generally configured in a distributed manner to achieve certain advantages that are provided by the architecture of the present invention. For example, in **Figure 5A**, configuration service **528** was briefly mentioned as supporting the configuration of the DKS framework, but no significant detail was provided.

The remaining description provides more detail on the manner in which the DKS framework can be configured to achieve certain benefits to solve certain issues within a multi-customer environment. Given a scenario in which a multi-customer service provider is using an integrated network management system for multiple customers, it is most likely that many different individuals will be assigned to manage different customers, different regions, and different groups of devices. In addition, separate individuals may have different duties within similar portions of the network, such as deploying new devices versus monitoring the uptime of those devices. In a highly distributed system comprising on the order of a million devices, the task of authenticating and authorizing the administrative actions of many individuals per customer, per region, per device, etc., becomes quite complex.

One significant issue with a large, multi-customer, network management framework is the manner in which the network management framework is to be configured and controlled. While the service provider may desire to perform most management operations for all of its customers, some customers may desire to control certain aspects of the on-going management process because they prefer to have more control over physical assets and enterprise data.

The network management framework of the present invention resolves these types of issues by allowing for a variety of dynamic, distributed, configuration operations during the management of the logical networks of multiple customers. The configuration is performed in a distributed manner that allows both a multi-customer service provider and its customers to have some control over the configuration and operation of the network management framework. In some cases, a customer may desire to have its own IT personnel perform certain duties within its IT infrastructure such that the IT personnel are directly responsible in certain ways to the management of the customer's organization. In these cases, a multi-customer service provider can configure major aspects of a customer's network management while other aspects can be controlled by the customer's own administrative personnel. The network management framework provides a security subsystem for controlling authentication and authorization such that customer administrators, i.e. individuals within a customer's own network administration department, are allowed to perform only certain operations.

To maximize the usefulness of the network management framework for a multi-customer service provider, the system provides for default configurations to be commonly deployed in multiple installations, thereby easing the service provider's burden and increasing the profitability in assuming control of one of its customer's network infrastructure. However, the configuration parameters may also be customized by the multi-customer service provider or by the customer. Specifically, the present invention preferably provides for multi-level, hierarchical configuration through inheritance with a local overriding capability for altering configuration values at the local level.

In particular, a default set of configuration parameters may be appropriate for many different customers in most aspects of the operation of the network management framework except for the configuration of ORB IDs for each customer. The overall network management components may be very similar across multiple customers except for the specific configuration of one or more scopes for each customer.

Because the present invention provides for distributed configuration, all configuration can be accomplished remotely, again saving effort and expenditures in managing a customer's infrastructure. In addition, the network management framework dynamically adapts to changes in configuration values; because the customer has a certain amount of control over configuration aspects of the system, components within the network management framework listen for such changes and dynamically adapt to new configuration parameters.

These features and other features are described in more detail below with reference to the remaining figures.

With reference now to **Figure 18A**, a block diagram shows possible administrative applications to be used by administrative users for controlling the network management framework by configuring components and reviewing status via the administrative applications. **Figure 18A** is similar to **Figure 5A**, which centers on the distributed IP driver subsystem by depicting the relationships between the distributed IP driver subsystem and various other components, such as the configuration service and administrative GUI applications. In **Figure 18A**, the administrative GUI applications represent the administrative interfaces to various services and DKS components.

IP driver administration application **1802** allows an administrative user to configure distributed IP drivers **1804**. Topology administration application **1806** allows an administrative user to configure topology service **1808**. IPOP administration application **1810** allows an administrative user to configure IPOP service **1812**, which interfaces to IPOP database **1814**. Service manager administration application **1816** may interface with other administrative applications and provides master control over configuring other services, as shown in **Figure 18B**.

With reference now to **Figure 18B**, a figure depicts a graphical user interface window that may be used by a network or system administrator to set the installation locations of various services. Window **1820** is a dialog

box for allowing an administrator to input configuration parameters for the service manager. Input fields **1822-1830** allow the administrator to specify the ORB ID of the ORB on which the administrator desires to install a particular service.

With reference now to **Figure 19**, a flowchart depicts a process by which a multi-customer administrator may complete the first stage of a multi-customer installation. The process begins with the multi-customer administrator configuring DKS components at installation time (step **1902**). In doing so, the multi-customer administrator might use many different GUI applications. For example, the multi-customer administrator may use a service manager administration application, such as the one shown in **Figure 18B**, to direct which services to start and the location of those services. The multi-customer administrator may also use various GUIs that are represented in the remaining figures that are discussed in more detail further below.

At some point, the multi-customer administrator would start the IPOP service on a specified ORB (step **1904**), and the multi-customer administrator would configure the IPOP service using a GUI application such as that shown in **Figure 20**. The multi-customer administrator then starts a number of IP driver instances on a set of ORBs to begin the dynamic discovery process (step **1906**). The number of IP drivers that would be started would be based on a variety of factors, such as: customer configuration; physical network attributes; boundaries, such as firewalls; slow links; approximate administrator estimate of the number of endpoints that

might be discovered and need to be monitored; geographic separation between networks; requirements for servicing a number of customers in a multi-customer environment; needs for multi-customer security, if there are common ORBs across customer boundaries; amount of distribution required based on the performance requirements for desired discovery time periods, desired monitoring time periods, and network action response times to be reflected within a topology administration application; or other administrative condition, concerns, or requirements.

After reviewing the status of various discovery and/or monitoring activities (step **1908**), the multi-customer administrator determines that the management of the multi-customer environment has reached a point at which the multi-customer administrator may allow management control to be commenced by customer administrators within the multi-customer environment (step **1910**). The process can then be considered complete, but it should be understood that the process may be repeated as often as necessary. The process shown in **Figure 19** may be performed when first installing a multi-customer environment, but the process is similar if a multi-customer environment is already operational and the multi-customer service provider desires to add another customer to the operational multi-customer environment.

With reference now to **Figure 20**, a figure depicts a graphical user interface window that may be used by a network or system administrator to configure some of the configuration parameters of the IPOP service. Window

2000 is presented by the IPOP administration application for configuring the IPOP service. Checkboxes **2002-2006** allow the administrator to select which types of database access will be supported by the network management framework. Input fields **2008-2010** allow an administrator to login to the system by specifying a user identifier and a password so that the system can authenticate and authorize the user for performing the IPOP configuration operations. Input field **2012** allow the user to specify URL information for the databases. Status values **2014-2016** report the number of endpoints that have already been discovered within a certain portion of the distributed data processing system and the number of IP drivers that have been deployed.

With reference now to **Figures 21A-21B**, a pair of flowcharts depicts a process by which a customer administrator within a multi-customer environment managed by the network management framework can perform certain management actions. The process shown in **Figures 21A-21B** show a configuration and management process that may occur during a second operational stage of operations within a multi-customer environment, whereas **Figure 19** depicted a first operational stage of operations within a multi-customer environment. The first stage process must be completed by a multi-customer administrator who configures certain aspects of the overall system, whereas the second stage process may be completed by either a multi-customer administrator or a customer administrator.

The process begins with the administrator creating additional IP drivers with correct customer anchor names (step **2102**) and additional IP drivers with correct

geographic anchor names (step **2104**). During the first operational stage, the multi-customer may configure certain aspects of the network management framework in a default manner, such as creating anchors with default names and parameters. Steps **2102-2104** show the customer customizing the installation process with more customer-specific configuration parameters in accordance with knowledge about the customer's IT infrastructure. Other customization steps that are not shown may be desired or required.

The administrator then checks the reported IP driver status to determine that enough of the endpoints have been discovered that other management actions may be performed (step **2106**). For example, the administrator may notice that an endpoint that was previously functioning properly during the discovery phase seems to have failed (step **2108**), and the administrator decides to investigate.

The administrator determines other DKS services that should be started to perform certain desired management actions for accomplishing the desired investigation of the suspect endpoint (step **2110**). The administrator starts the NEL/Gateway services on a particular ORB (step **2112**) followed by starting the topology service on the ORB (step **2114**). The administrator views the topology of the portion of the distributed data processing system that is of interest because it includes the suspect endpoint (step **2116**). The administrator then selects the suspect endpoint in the topology administration application and requests a management action on the suspect endpoint through the appropriate GUI (step **2118**).

The topology administration application asks IP mapper to perform the management action on the suspect endpoint (step **2120**), and IP mapper requests an AAO from the NEL/gateway services for the requested management operation (step **2122**). The NEL service gets the routing information (step **2124**) and returns an AAO to IP mapper (step **2126**). IP mapper then uses the returned AAO to perform the management action and to return status back to topology administration application (step **2128**). The topology administration application then displays the results of the requested management operation to the administrator (step **2130**), after which the process is complete. The administrator may review the status and request other operations as desired.

As distributed computing and the reliance on large and sophisticated networks become commonplace in business, the need to discover and monitor the devices in the network is of utmost importance. The network discovery/monitor software must be suitable for very large-scale and multi-customer environments; as such, it must be scalable, flexible, reliable and secure. The configuration and runtime manipulation of this type of network discovery/monitoring service is a complicated task; the following figures describe a graphical user interface to facilitate the administration of the DKS network management framework.

There are two main administrative tasks to be done for the distributed network discovery and monitor system described above: configuration and status checking. **Figures 22-29** described below depict various aspects of configuration.

Configuration: As described above, the network discovery/monitor service (IP driver service) used in a large multi-customer environment must be highly configurable. The administrator must be able to define the number of distributed monitors needed, where they are located, and specific properties that will direct the behavior of each monitor. The configurable properties can include, but are not limited to:

(1) General properties, such as IP driver ID, which is a numeric identifier to distinguish an IP driver from all the others, and number of threads, which are the number of threads available to perform network discovery and polling tasks;

(2) Scope, which is one or more subnetworks that a given IP driver is responsible for discovering and monitoring;

(3) Discovery mechanisms, which are the ways a particular IP driver can discover systems within its scope and may include, but are not limited to, ping spread discovery, route table poll, ARP table poll, unsolicited ping discovery, or any combination of the above;

(4) General node information, such as: (a) polling strategy, which is the way a given node will be monitored (if any)--pings may be used to get general up/down status and SNMP queries may be used for more extensive systems (the administrator may also wish to ignore i.e. not monitor, any node that is not running an SNMP agent); and (b) delete interval, which is the time to wait before a non-responding address is deleted from the data store of network objects;

(5) DHCP node information, such as: (a) address range, which is the address ranges that an administrator knows to be assigned dynamically through the Dynamic Host Configuration Protocol (DHCP)--hosts with these addresses are often transient and may be treated differently from more permanent hosts that have static addresses, e.g., these nodes may be deleted from the network data if they do not respond within a shorter timeframe than non-DHCP nodes; and (b) delete interval, which is the time to wait before a non-responding DHCP address is deleted from the data store of network objects.

The figures below illustrate one possible layout for the presentation and manipulation of the configuration data.

With reference now to **Figure 22**, a GUI window is shown that might be used for input and viewing of general configuration properties. Window **2200** allows a user to provide input into fields for entering the IP driver ID and number of threads assigned to a particular monitor. The action buttons at the bottom of the window ("OK", "Apply", "Undo" and "Cancel") may be assigned appropriate semantics. For example, the "OK" button could mean "apply these values and return to the previous window" or it could mean "save these values and apply them whenever the 'Apply' button is pushed, in this or another window".

With reference now to **Figure 23**, a GUI window is shown that might be used for input and viewing of scope properties. The user is provided with window **2300** containing input fields to enter the subnet(s) that a particular monitor will detect, along with the subnet's associated mask, priority, optional customer ID and

optional private network ID. Rows may also be added by dragging and dropping visual icons that represent subnets from associated network topology GUIs onto the window area displaying the scope information. Rows may be removed by highlighting and then hitting the "Delete" key or by right-clicking and selecting "Cut" from the action menu. The action buttons at the bottom of the window are the same as described for **Figure 22**.

With reference now to **Figure 24**, a GUI window is shown that might be used for input and viewing of discovery mechanism properties. The user is provided with window **2400** containing check boxes to select or deselect the specific discovery mechanism(s) a particular monitor should use. Input boxes are provided for the administrator to input "seed" addresses--those at which the discovery processes will start. After entering an address, the user selects the "Add" button. Seeds may also be added by dragging and dropping visual icons that represent system nodes or interfaces from associated network topology GUIs onto the window area displaying the seed.

With reference now to **Figure 25**, a GUI window is shown that might be used for input and viewing of ARP table discovery properties. The user is provided with window **2500** containing input fields to enter the interval at which to poll the ARP tables and the maximum number of entries to be read.

With reference now to **Figure 26**, a GUI window is shown that might be used for input and viewing of routing table discovery properties. The user is provided with window **2600** containing input fields to enter the interval

at which to poll the routing tables and the maximum number of entries to be read from one table. This window would be non-selectable if the routing table discovery mechanism is not selected as a discovery type (see **Figure 24**). The action buttons at the bottom of the window are as described for **Figure 22**.

With reference now to **Figure 27**, a GUI window is shown that might be used for input and viewing of ping spread discovery properties. The user is provided with window **2700** containing input fields to enter the interval at which to execute ping spread operations, an IP address mask to limit the machines that are actually contacted and the time interval to wait between individual pings. This window would be non-selectable if the ping spread discovery mechanism is not selected as a discovery type (see **Figure 24**). The action buttons at the bottom of the window are as described for **Figure 22**.

With reference now to **Figure 28**, a GUI window is shown that might be used for input and viewing of general node properties. The user is provided with window **2800** containing check boxes to indicate whether ping and/or SNMP queries should be used to monitor the nodes, as well as a check box to indicate if nodes which are not running an SNMP agent should be monitored at all. An input field is provided for defining the interval after which non-responding nodes should be deleted from the internal data store. Window **2800** could also provide an input field for a network address, so that the configuration data supplied might be applied specifically to that one address, but not in general; the general case would be

indicated by a blank address field. The action buttons at the bottom of the panel are as described for **Figure 22**.

With reference now to **Figure 29**, a GUI window is shown that might be used for input and viewing of DHCP node properties. Window **2900** contains input boxes for the administrator to input DHCP addresses or address ranges--those addresses that are assigned to systems dynamically through the Dynamic Host Configuration Protocol. After entering an address, the user selects the "Add" button. DHCP addresses may also be added by dragging and dropping visual icons that represent DHCP systems from associated network topology GUIs onto the window area displaying the DHCP addresses. Rows may be removed by highlighting and then hitting the "Delete" key or by right-clicking and selecting "Cut" from the action menu, or they may be deleted by highlighting an address and selecting the "Delete" button. An input field is provided for defining the interval after which non-responding DHCP nodes should be deleted from the internal data store. The action buttons at the bottom of the window are as described for **Figure 22**.

As noted above, there are two main administrative tasks to be done for the distributed network discovery and monitor system described above: configuration and status checking. **Figures 30-33** described below depict various aspects of status checking.

Status Checking: In the course of operation, the administrator may want to review or change the configured parameters of a distributed network monitor controller or may want to view the tasks currently scheduled or

executing in the monitor. The status checking operations may include, but are not limited to:

(1) Configuration Status, which shows all of the current configuration parameters for this monitor;

(2) Thread Status, which shows the state of all of the polling threads in this monitor; and

(3) Task Status, which shows the state the discovery/monitor tasks currently scheduled or running.

The figures below illustrate one possible layout for the presentation and manipulation of status data.

With reference now to **Figure 30**, a GUI window is shown that might be used for reviewing configuration information for a given monitor. In window **3000**, selecting any field would navigate the administrator to the configuration window for that item where, if allowable, the properties might be changed and applied to the running software. The action buttons at the bottom of the window are as described for **Figure 22**.

With reference now to **Figure 31**, a GUI window is shown that might be used for reviewing the status of the polling threads in a given monitor. In window **3100**, by highlighting a row and right-clicking the mouse, the administrator could be presented a menu of actions to be performed on the thread, such as delete, stop, run or change priority. If any thread is deleted, the "number of threads" configuration property is automatically updated. The action buttons at the bottom of the window are as described for **Figure 22**.

With reference now to **Figure 32**, a GUI window is shown that might be used for reviewing the status of the network discovery/monitoring tasks in a given monitor.

In window **3200**, by highlighting a row and right-clicking the mouse, the administrator could be presented a menu of actions to be performed on the thread which is assigned to the task, such as delete, stop, run or change priority. If any thread is deleted, the "number of threads" configuration property is automatically updated. The action buttons at the bottom of the window are as described for **Figure 22**.

With reference now to **Figure 33**, a GUI window shows one possible layout for a navigation window. In addition to the windows described above, a navigation feature as shown in window **3300** can be added to the interface (perhaps in a frame separate from the configuration and status windows, but visible at the same time). This navigation tool can be used to move easily from one window to another.

Although network information discovered by the IP driver subsystem can be obtained by use of a simple command line interface, most of today's software products require a GUI interface to promote ease of use. The topology service has been made available for displaying objects and relationships to users; however, this service is somewhat "generic" in that it is available for a multitude of services to utilize. Therefore, a mechanism is necessary for interpreting what the IP driver service discovers and telling topology how to display this information. This mechanism is referred to as IP mapper.

The major advantage of the IP mapper is that it allows the topology service to remain "generic"; topology remains a service for a variety of components to use rather than having to contain IP driver-specific logic.

The topology service provides four types of objects to its customers: Aggregate Topology Objects, Topology Objects, Resources, and Relationships. IP Mapper allows the topology service to work independently of IP driver through the following means. First of all, IP mapper creates its own resources to use, so that the topology service toes not have to provide resource types. Resources and their associated data fields are declared in an XML file. IP Mapper declares four types of resources: networks, systems, routers, and endpoints. Each created resource dictates the image displayed to represent the resource as well as what data is contained by a resource. Secondly, IP mapper creates Aggregate Topology Objects to represent each network, system, or router that IP driver discovers. Relationships are then formed between the systems, networks, and routers in which they reside. Third, discovered endpoints are represented by Topology Objects, and relationships are created between the endpoints and the systems and routers they reside in. Finally, if a system or router contains endpoints that reside in separate networks, virtual resources must be created. Virtual resources reside inside a network, and appear just as a system or router would but actually link back to the original router or system so that data is not duplicated and the rules of which administrators can administrate objects managed by a particular IP driver can be enforced.

The graphical user interface described above is tailored to the features required by a modern network discovery/monitoring system. The present invention is logically composed of several distributed entities, each

of which is independently configurable and reviewable.

The advantages of the present invention should be apparent in view of the detailed description of the invention that is provided above. A network management framework flexibly manages multiple customers within a highly distributed system. The network management framework allows for a variety of dynamic reconfigurations of logical networks for multiple customers. The configuration is performed in a distributed manner that allows both a multi-customer service provider and its customers to have some control over the configuration and operation of the network management framework.

The distributed nature of this system allows the network to be partitioned into manageable scopes, so that any one IP driver in the system is not overloaded. The data is cached only to configured memory limits and the bulk of the data is stored and fetched from persistent storage. These factors enable a distributed monitoring system to handle very large networks (on the order of millions of devices). As a comparison, IBM's NetView product is limited to approximately 20,000 devices. In the system described, configuration data is stored in a central location with a small amount of overriding system-specific residing locally, allowing for easy administration of many independently functioning monitors.

The distributed nature of this system allows malfunctions to be limited to a fraction of the total monitored network. Failover capability in the encompassing system assure quick restart of a failed

network monitor, even if it means moving the functionality to a different machine.

Configuration parameters allow each IP driver in the monitoring system to have different behavior, suited to the particular machine it runs on and the network it is operating on. A variety of discovery mechanisms can be enabled or disabled to extract the proper amount of network data. Implementation in a platform-neutral language such as Java makes it possible to run IP drivers on numerous hardware and operating systems.

Access control applied at the object level (DKS framework, IP driver instance, network, machine, interface endpoint) guarantees that no one can access that data other than the user it is intended for. This is especially important in a multi-customer environment.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of instructions in a computer readable medium and a variety of other forms, regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include media such as EPROM, ROM, tape, paper, floppy disc, hard disk drive, RAM, and CD-ROMs and transmission-type media, such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration but is not intended to be exhaustive or limited to the disclosed

embodiments. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiments were chosen to explain the principles of the invention and its practical applications and to enable others of ordinary skill in the art to understand the invention in order to implement various embodiments with various modifications as might be suited to other contemplated uses.